Александр Климов

Занимательное программирование на Visual Basic .NET

Санкт-Петербург
«БХВ-Петербург»
2005

УДК 681.3.068+800.92VB.NET
ББК 32.973.26-018.1
К49

Климов А. П.

К49 Занимательное программирование на Visual Basic. NET. — СПб.: БХВ-Петербург, 2005. — 528 с.: ил.

ISBN 5-94157-572-6

На занимательных примерах рассмотрено программирование на языке Visual Basic .NET. Описана работа с текстами и строками: от создания бегущих строк и мигающих заголовков до программирования различных текстовых эффектов. Рассмотрены примеры создания геометрических фигур и рисование различных кривых линий (от синусоиды до спирали Архимеда), а также примеры программирования градиентных заливок, геометрического преобразования объектов и сложных фигур: звезд, фигуры Инь-Янь, снежинок и др. Описана работа с импортируемыми изображениями: вращение, буксировка, наложение, плавное появление одной картинки из другой, куб с картинками на гранях и др. Рассмотрены примеры, связанные с работой клавиатуры и мыши. Показаны интересные особенности использования форм и элементов управления среды Visual Basic .NET. Приведены примеры создания игровых, шуточных, музыкальных и говорящих программ, а также различные полезные трюки и приемы программирования.

Книга сопровождается компакт-диском, содержащим все описанные примеры.

Для программистов

УДК 681.3.068+800.92VB.NET
ББК 32.973.26-018.1

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Игорь Шишигин*
Зав. редакцией	*Григорий Добин*
Компьютерная верстка	*Натальи Смирновой*
Корректор	*Наталия Першакова*
Дизайн обложки	*Игоря Цырульникова*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 25.04.05.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 42,57.
Тираж 3000 экз. Заказ №
"БХВ-Петербург", 194354, Санкт-Петербург, ул. Есенина, 5Б.

Санитарно-эпидемиологическое заключение на продукцию № 77.99.02.953.Д.006421.11.04
от 11.11.2004 г. выдано Федеральной службой по надзору
в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 5-94157-572-6

Оглавление

Введение

Дорогой читатель! Если вы купили эту книгу, надееясь найти на ее страницах примеры работы с базами данных, криптографией, офисными приложениями и тому подобными серьезными проектами, то сделали большую ошибку. Пойдите в магазин и сдайте книгу назад. Безусловно, работа с базами данных — это очень важно, а вы — весьма деловой человек, но моя книга совсем о другом. Здесь рассматривается только *занимательное* программирование.

О чем эта книга

Книги, посвященные освоению любого языка программирования, как правило, слишком серьезны и строги. Такой подход оправдан при изучении языка студентами для получения теоретических основ. Но для людей, самостоятельно изучающих язык, а также тех, кто хочет расширить свои познания, гораздо лучше изучать язык по другой методике. Общеизвестно, что даже самую скучную и нудную работу легче делать, если подойти к ней творчески. К счастью, тенденция меняется в лучшую сторону. В продаже появляются книги, в которых рассматриваются не абстрактные примеры, а интересные задачи. Вот и в этой книге я сделал попытку взглянуть на изучение Visual Basic .NET с другой стороны. Здесь собраны примеры, которые помогут лучше понять особенности языка через игры, шутки и создание красивых узоров и эффектов.

Для кого эта книга

В первую очередь книга предназначена для программистов средней и высокой квалификации. Здесь не будут объясняться азы программирования — как запустить программу, создать проект и т. д. Впрочем, начинающий программист тоже может воспользоваться этой книгой, так как я постарался подробно разобрать наиболее сложные моменты программирования. Очень хорошо, если читатель уже имел опыт программирования на Visual Basic 6.0, так как многие примеры переделаны из старых проектов. И на страницах книги мы не раз обратимся к особенностям перехода от Visual Basic 6.0 к Visual Basic .NET.

Требования, предъявляемые к читателю

Все примеры, приведенные в книге, написаны на Visual Basic .NET 2003. Но с вероятностью в 99,9 процента можно утверждать, что примеры будут работать и в Visual Basic 2002, а также в новых реализациях Visual Basic .NET, так как в примерах используются основополагающие принципы программирования, которые не подвержены кардинальным изменениям.

Дополнительные сведения

Выход новой версии любого языка программирования — это для программиста всегда головная боль. Приходится заново изучать новые возможности, предоставляемые последней версией программы. Не стал исключением и выход Visual Basic .NET. Если изменения в VB 6.0 по сравнению с VB 5.0 носили, скорее, косметический характер, то переход на версию VB .NET называют не иначе как революцией.

Анекдот в тему

Перепись населения у программиста:

– Ваш родной язык.

– Как это родной язык?

– Ну какой Вы язык с детства изучали, всю жизнь использовали?

– Basic.

– Да нет, настоящий.

– А! Настоящий! Тогда Си.

Теперь это в прошлом! Visual Basic .NET, использующий все возможности платформы .NET Framework, теперь такой же полноценный язык, как C# или C++. В новой версии VB .NET изменилось почти все — появились новые функции, методы, объекты, стиль программирования. А что делать программистам со своим багажом примеров, которыми они активно пользовались? Изучать новый язык и переделывать примеры. Вот и я столкнулся с такой проблемой. Пришлось заново пересматривать свою коллекцию исходных кодов. При написании книги я использовал примеры из своей обширной библиотеки, расположенной на сайте **http://rusproject.narod.ru/**

Основная масса примеров была написана на VB 6.0. Мне пришлось переписывать коды к примерам практически заново. Также я использовал различные источники из Интернета и книг. По ходу повествования я буду указы-

вать на эти источники, в которых вы можете почерпнуть немало дополнительных интересных сведений, также им будет посвящено *Заключение*.

Следует отметить, что приводимые примеры не являются верхом совершенства. Дело в том, что я стал собирать свою коллекцию примеров, когда сам только начинал осваивать премудрости программирования. Теперь, спустя несколько лет, я просматриваю исходные коды и замечаю, что где-то можно оптимизировать, где-то переписать заново, где-то просто удалить лишнее. Но я решил не менять радикально эти примеры. И улучшение примеров пусть останется вам в качестве домашнего задания. И мой совет — не пытайтесь механически переписывать пример. Постарайтесь придумать свой вариант, поиграть с различными параметрами. В общем, активно изучайте язык Visual Basic .NET. Только в этом случае можно рассчитывать на полное и уверенное овладение языком программирования. А я в меру своих скромных сил постараюсь вам помочь.

Как пользоваться примерами

На прилагаемом компакт-диске вы сможете найти все примеры, которые приводятся в книге. Примеры разбиты в папках по номерам глав. Например, в папке 7 находятся примеры из седьмой главы. Внутри каждой такой папки ищите файл с расширением sln. Запустив этот файл в Visual Basic .NET 2003 (или Visual Studio .NET 2003), вы загружаете тем самым все проекты выбранной главы. Чтобы запустить конкретный проект, вам необходимо выделить название проекта и в контекстном меню выбрать команду **Set as StartUp Project**.

Благодарности

Написание книги отняло у меня много сил. Мне пришлось долго и тщательно отбирать примеры, которые могли бы показаться вам интересными и занимательными. Кроме того, необходимо было найти золотую середину в стиле повествования — хотелось, чтобы книга выглядела не только как тривиальный сборник примеров, но и стала для вас увлекательным чтением. Моя личная жизнь отошла на второй план. У меня почти не оставалось времени ни на ремонт квартиры, ни на встречи с друзьями...

Существует множество сайтов по программированию, на которые есть возможность присылать свои примеры для ознакомления. Однажды кто-то назвал такие сайты свалкой файлов. Я категорически против такого подхода. Эти сайты можно сравнить с залежами полезных ископаемых. И, скачивая мегабайты кода на свой компьютер, находишь яркие и запоминающиеся проекты, изучение которых доставляет удовольствие. К сожалению, я не

могу назвать поименно всех авторов примеров, которые скопились в моей коллекции. Количество скачанных и изученных файлов не поддается никакому учету. Но всем им Большое спасибо!

Отдельно хочется выразить свою признательность руководителю проекта Игорю Шишигину и редактору Григорию Добину, которые помогли мне написать данную книгу.

Кот Рыжик любезно разрешил использовать фотографии со своим изображением в качестве иллюстраций к некоторым примерам при условии, что я угощу его удвоенной порцией любимой им печенки. Что я обязательно сделаю.

Александр Климов

Москва, 2005 год

ЧАСТЬ I

Строки и тексты

Глава 1

Эффекты со строками

Примеры научают лучше,
нежели толкования и книги.

Н. И. Лобачевский

А начну я свою книгу с самого простого — со строк. Казалось бы, что можно сделать со строками? И, тем не менее, можно придумать несколько интересных эффектов. Прежде всего, надо помнить, что строка состоит из отдельных элементов — символов. Можно, используя возможности таймера, выводить требуемую строку не целиком, а по мере надобности. Действуя так, мы получим анимационные эффекты печатающейся строчки, бегущей строки и т. д. Давайте перейдем к реализации наших планов.

1.1. Печатающаяся строка

Создадим новый проект `TypingText` (листинг 1.1). Расположим на форме надпись `Label1`, две кнопки `Button1` и `Button2`, а также таймер `Timer1`.

Совет

В наших примерах мы часто будем оставлять имена элементов управления по умолчанию. В реальных же проектах настоятельно рекомендуется использовать говорящие имена, например для кнопки можно использовать имя `butAniString` (Анимированная строка).

Свойству `Text` метки `Label1` присвоим значение `Занимательное программирование`, свойству `Text` кнопки `Button1` — значение `Старт`, а свойству `Interval` таймера `Timer1` присвоим значение `500`. У кнопки `Button2` присвоим свойству `Text` значение `Кнопка`. Можете добавить код по своему усмотрению для обработки щелчка данной кнопки. Свойству `Text` нашей формы `Form1` мы присвоим значение `Печатающаяся строка`. Данный текст будет отображаться в заголовке формы, указывая на цель наших упражнений. Остальные свойства оставляем по умолчанию. Теперь дважды щелкаем на

кнопке Button1, чтобы написать код, который будет выполняться при щелчке мыши на кнопке.

Листинг 1.1. Пример печатающейся строки

```
'----------------------------------------
' Печатающаяся строка © 2004 Александр Климов
'----------------------------------------
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
        If Button1.Text = "Старт" Then
            Timer1.Enabled = True
            Button1.Text = "Стоп"
        Else
            Timer1.Enabled = False
            Button1.Text = "Старт"
        End If
    End Sub


    Private Sub Timer1_Tick(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Timer1.Tick
        ' Анимация слева направо
        Dim title As String = "Занимательное программирование"
        Static c As Integer = 0
        Label1.Text = title.Substring(0, c)
        Me.Text = title.Substring(0, c)
        c += 1
        If c = title.Length() + 1 Then c = 0

        ' Анимация справа налево
        Dim title2 As String = "Кнопка"
        Static i As Integer = title2.Length()
        Button2.Text = title2.Substring(0, i)
        i -= 1
        If i = -1 Then i = title2.Length()
    End Sub
```

Разберем, что мы написали. Кнопка с надписью **Старт** запускает таймер, при этом надпись **Старт** поменяется на **Стоп**. При повторном нажатии надпись **Стоп** заменяется обратно на **Старт**. При этом происходит включение и

выключение таймера. Вся работа по созданию эффекта происходит в событии таймера `Tick`. Далее идет объявление строковой переменной `title`, которой мы присвоили сразу же значение `Занимательное программирование`. Затем мы создаем статическую переменную `c`, которая служит счетчиком (counter). С помощью метода `Substring` класса `String` мы как бы наращиваем строку, начиная с первого символа и заканчивая последним. После чего обнуляем счетчик и все повторяем снова. Как видите, ничего сложного в реализации эффекта нет, а результат получился достаточно интересным. Вы можете в любой момент прервать анимацию строки, а повторным нажатием на кнопку возобновить вывод строки с остановленной позиции. Чтобы пример выглядел более интересным, я добавил анимированную строку и в заголовок формы:

```
Me.Text = title.Substring(0, c)
```

Если вам хочется не наращивать текст, а наоборот, уменьшать его, то данный пример легко переделать. В этом случае придется создать еще один счетчик `i`. Начальное значение счетчика равно длине заданной строки. При каждом срабатывании таймера будет вычитаться один символ из строки. При достижении счетчиком значения 0 снова присвоим ему старое значение длины строки и применим данный код не к метке, а ко второй кнопке. Ну, вот, мы и написали первый простой пример, который показывает, что и простыми методами можно добиваться интересных эффектов. Далее мы разберем более сложные примеры.

1.2. Мигающий заголовок

Усложним немного пример. Главной его особенностью будет отказ от использования визуальных средств при создании таймера. Умение программно создавать объекты, не прибегая к помощи панели инструментов, полезно при создании проектов без помощи пакета Visual Studio. Да-да, создавать программы можно даже при помощи обычного Блокнота.

Примечание

В состав .NET Framework входит бесплатный компилятор проектов. Написание кода в Блокноте тоже достаточно занимательное занятие, хотя, наверное, здесь больше подходит термин *экстремальное программирование*. Но в некоторых случаях это умение может быть востребовано. Например, у вас есть доступ к компьютеру с установленным .NET Framework, но без пакета Visual Studio .NET. Тогда вам ничего не остается, как написать программу в текстовом редакторе.

Но перейдем к примеру. Мы немного видоизменим способ анимации текста в заголовке формы. Теперь строка будет не только увеличиваться, но еще и мигнет несколько раз, привлекая внимание пользователя. Данный пример

взят из моей коллекции примеров, написанных на VB 6.0, и переделан с учетом новых возможностей VB .NET 2003. Откроем новый проект и назовем его `FlashText`. Вставим следующий код в редакторе кода (листинг 1.2).

Листинг 1.2. Мигающая анимированная строка

```
'---------------------------------------
' Мигающая анимированная строка © 2004 Александр Климов
'---------------------------------------
    Dim title As String = "Занимательное программирование"
    Dim b As Integer
    Dim p As Integer
    Dim I As Integer

    'Создадим два таймера программным путем
    Dim tmr As New System.Timers.Timer
    Dim tmr2 As New System.Timers.Timer

    Private Sub Form1_Load(ByVal sender As Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        AddHandler tmr.Elapsed, AddressOf OnTimedEvent
        tmr.Interval = 200
        tmr.Enabled = True
        AddHandler tmr2.Elapsed, AddressOf OnTimedEvent2
        tmr2.Interval = 100
        tmr2.Enabled = False
        CheckAgain()
    End Sub

    Public Sub OnTimedEvent(ByVal source As Object, ByVal e As
System.Timers.ElapsedEventArgs)
        Dim t As String
        t = Microsoft.VisualBasic.Left(title, b)
        MyClass.Text = t
        b = b + 1
        If b > I Then
            b = 0
            tmr.Enabled = False
            tmr2.Enabled = True
```

```
            p = 0
        End If
    End Sub

   Public Sub OnTimedEvent2(ByVal source As Object, ByVal e As
System.Timers.ElapsedEventArgs)
        p = p + 1
        If p Mod 2 = 0 Then
            MyClass.Text = title
        Else
            MyClass.Text = ""
        End If
        If p = 10 Then
            tmr2.Enabled = False
            tmr.Enabled = True
        End If
    End Sub

    Sub CheckAgain()
        I = Len(title)
        b = 0
        p = 0
    End Sub
```

Разберем написанный код. Сначала мы объявили на уровне класса несколько переменных и два таймера. Обратите внимание, что мы не пользовались услугами визуального редактора, чтобы использовать элементы управления (в данном случае таймеры) в своем проекте. При загрузке формы таймеры включаются путем добавления двух обработчиков событий `Elapsed`. Также при загрузке формы я присвоил этим таймерам некоторые свойства и включил процедуру `CheckAgain`, о которой речь впереди. Для первого таймера обработчик события достаточно простой и практически совпадает с кодом предыдущего примера с самопечатающей строкой. Единственное отличие — мы подключаем второй таймер, который обеспечивает мигание заголовка формы. Процедура `CheckAgain` просто устанавливает старое значение для длины заголовка и обнуляет значения переменных `b` и `p`. Как видите, на самом деле сложного тут ничего нет. Сложным я назвал этот пример потому, что техника использования таймеров программным путем и написание обработчика событий поначалу вызывают некоторые неудобства у программистов, которые только недавно начали переходить от VB 6.0 к VB .NET.

1.3. Бегущая строка

Вероятно, вам приходилось видеть бегущую строку на электронных табло, где периодически повторяется одно и то же рекламное сообщение. Попробуем воспроизвести данный эффект в проекте `RunTitle` (листинг 1.3). Для примера достаточно поместить на форму только таймер.

Листинг 1.3. Бегущая строка

```
'--------------------------------------
' Бегущая строка © 2004 Александр Климов
'--------------------------------------
    Private Sub Timer1_Tick(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Timer1.Tick
        Dim StrTemp As String
        StrTemp = Me.Text
        StrTemp =  Microsoft.VisualBasic.Right(_
      StrTemp, Len(StrTemp) - 1) & Microsoft.VisualBasic.Left(StrTemp, 1)
        Me.Text = StrTemp
    End Sub

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As Sys-
tem.EventArgs) Handles MyBase.Load
        ' Необходимо добавить несколько пробелов,
        ' чтобы строки не слипались
        Me.Text = "Занимательное программирование       "
    End Sub
```

В этом примере мы отбрасываем первый символ из строки и добавляем его в конец нашей строки. Чтобы создать пространство между бегущими строками, пришлось добавить несколько пробелов в выводимый текст.

1.4. Эффект "Матрицы"

Вот еще один эффект со строкой, который я назвал *эффектом "Матрицы"*. Тот, кто смотрел этот фильм, помнит мелькающие буквы в его начале. Я попробовал реализовать этот эффект в проекте `Matrix` (листинг 1.4). Для примера понадобится таймер `Timer1` с интервалом, равным `100`, и свойством `Enable = True`.

Листинг 1.4. Эффект "Матрицы"

```
'--------------------------------------
' Эффект "Матрицы" © 2004 Александр Климов
'--------------------------------------
    Dim FXCounter As Integer = 0
    Dim FXCounter2 As Integer
    Dim SingleChr As String

    Private Sub Timer1_Tick(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Timer1.Tick

        ' Образец - текст, который должен появиться в конечном итоге
        Dim StrTemp As String
        StrTemp = "MATRIX"

        ' Проходим в цикле по всем символам
        ' Справа налево
        If FXCounter = 0 Then
            FXCounter2 = 31
            FXCounter = 1
        End If

        ' Позиция для замены символа
        Dim iStart As Integer
        iStart = Len(StrTemp) - (FXCounter - 1)

        Try
            If FXCounter2 = 31 Then SingleChr = Mid$(StrTemp, iStart, 1)
            FXCounter2 = FXCounter2 + 1
            Mid(Me.Text, iStart, 1) = Chr(FXCounter2)
            If FXCounter2 = Asc(SingleChr) Then
                FXCounter2 = 31
                FXCounter = FXCounter + 1
            End If

        Catch ex As Exception When iStart = 0
            'Обрабатываем ошибку
            Timer1.Enabled = False
```

```
        End Try
    End Sub

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        Me.Text = "Kitten"
    End Sub
```

1.5. Заключение

В примерах этой главы мы рассматривали строки как набор символов. Но можно ведь рассматривать их как графические объекты. В этом случае мы можем разукрасить буквы разными цветами, задать градиентную заливку, вращать буквы вокруг своей оси и вдоль какой-нибудь кривой. Но все это будет рассмотрено в следующих главах.

1.6. Советы и рекомендации

Попробуйте придумать новые собственные эффекты для строк. Вот несколько рекомендаций.

- Измените пример из *разд. 1.3* таким образом, чтобы бегущая строка бежала слева направо.
- В этом же примере из *разд. 1.3* используются старые функции `Left` и `Right`, оставшиеся от VB 6.0 в целях совместимости. Попробуйте переписать пример с использованием методов класса `System.String`.

Глава 2

Игры с текстами

В этой главе мы поговорим о том, что можно сделать с текстами. Читатель спросит: "А чем, собственно, отличаются строки от текстов?" Вопрос справедливый — деление на строки и тексты в данном случае условно. Просто мне хотелось разъединить две области применения текстов в программе. Операции, которые мы проводили в *главе 1*, можно использовать, например, в заголовках формы. Согласитесь, что в заголовке формы мы не можем менять размеры шрифтов, цвет символов или вращать символы в разных направлениях. Теперь же у нас уже больше возможностей для создания красивых эффектов с использованием разных размеров символов и цветов.

2.1. Использование узорных и градиентных кистей

Самый простой способ добиться некоторого эффекта при работе с текстом — это использование штриховых и градиентных кистей. Более подробно о кистях я расскажу в *главе 3*, посвященной графике, а пока приведу небольшой пример — проект `UsingPens` (листинг 2.1).

Листинг 2.1. Использование узорных и градиентных кистей

```
'---------------------------------------
' Использование кистей © 2004 Александр Климов
'---------------------------------------
Imports System.Drawing.Drawing2D
Dim f As Font = New Font("Tahoma", 72)
Dim sBrick As String = "Кирпичики"
Dim sGradient As String = "Градиент"

    Private Sub Form1_Paint(ByVal sender As Object, ByVal e As
System.Windows.Forms.PaintEventArgs) Handles MyBase.Paint
        Dim g As Graphics = e.Graphics
```

```
        ' Создаем кисть с узором кирпича
        Dim hbr As New HatchBrush(HatchStyle.HorizontalBrick, _
            Color.White, Color.Tomato)
        ' Выводим строку, закрашенную кирпичами
        g.DrawString(sBrick, f, hbr, 0, 0)

        Dim rect As New Rectangle(10, 50, _
                     ClientSize.Width, ClientSize.Height)
        ' Создаем градиентную кисть
        Dim lgrb As New LinearGradientBrush(rect, _
                     Color.Violet, Color.SkyBlue, _
                     LinearGradientMode.BackwardDiagonal)
        g.DrawString(sGradient, f, lgrb, 0, 100)

    End Sub
```

В этом примере я применяю две разновидности кистей: первая кисть использует узор кирпича, вторая — градиентную заливку. Результат программы представлен на рис. 2.1.

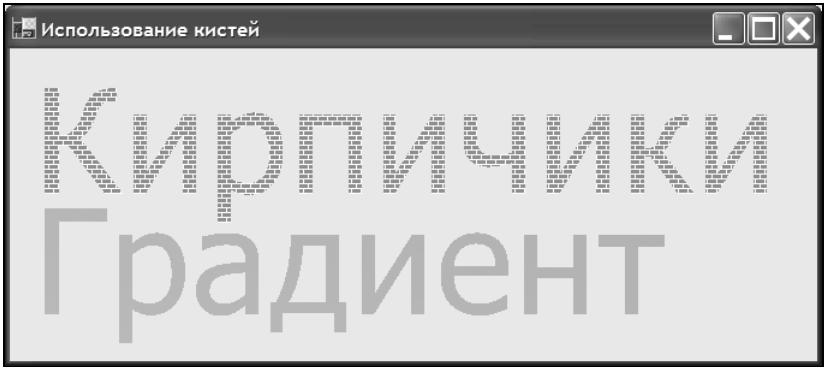

Рис. 2.1. Использование кистей для текстов

2.2. Объемный текст

Зачастую на форму выводится обычный текст, который выглядит не слишком красиво. Но можно придать текстам объемность. Причем существуют два эффекта объемности — выпуклый и вдавленный вид. Достигаются подобные эффекты простым смещением заданного текста с другим цветом,

имитирующим тень. Этот способ широко распространен и с успехом используется в самых различных областях. Но пора переходить к практике. Проект назовем `3DText` и поместим на форму одну кнопку. Для начала реализуем эффект выпуклости. Для этого нам понадобятся два вызова метода `DrawString`. При первом вызове метода на форме рисуется заданная строка серым цветом, при втором вызове — строка белого цвета со смещением в 1 или 2 пиксела (листинг 2.2).

Листинг 2.2. Создание выпуклого текста

```
'--------------------------------------
' Объемный текст © 2004 Александр Климов
'--------------------------------------
     Private Sub Form1_Paint(ByVal sender As Object, ByVal e As
System.Windows.Forms.PaintEventArgs) Handles MyBase.Paint
        Dim sb As New SolidBrush(Color.Gray)
        Dim f As New Font("Tahoma", 48, FontStyle.Bold)
        Dim g As Graphics = e.Graphics
        g.DrawString("Котенок", f, sb, 10, 10)
        sb.Color = Color.White
        g.DrawString("Котенок", f, sb, 8, 8)
    End Sub
```

Запустите проект и убедитесь, что выводимый текст выглядит выпукло (рис. 2.2).

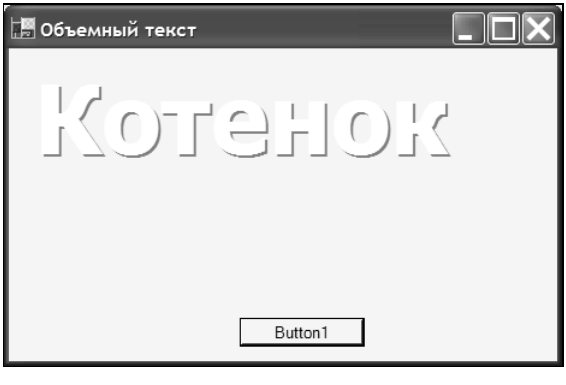

Рис. 2.2. Выпуклый текст

Для эффекта вдавленного текста необходимо сместить вторую строку ниже первой. Этот пример реализован через кнопку `Button1` (листинг 2.3).

Листинг 2.3. Создание вдавленного текста

```
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
        Dim sb As New SolidBrush(Color.Gray)
        Dim f As New Font("Tahoma", 48, FontStyle.Bold)
        Dim g As Graphics = CreateGraphics()
        g.Clear(BackColor)
        g.DrawString("Котенок", f, sb, 10, 10)
        sb.Color = Color.White
        g.DrawString("Котенок", f, sb, 12, 12)
        g.Dispose()
    End Sub
```

Снова запустите проект и нажмите на кнопку. Вы должны увидеть, что текст теперь стал вдавленным (рис. 2.3).

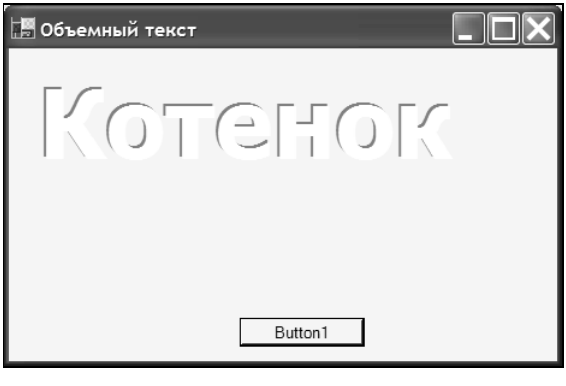

Рис. 2.3. Вдавленный текст

2.3. Контурный текст

Текст, выводимый на экран, по умолчанию заполнен каким-то цветом. Но если вам нужно вывести только контур символов, то можно воспользоваться классом `DrawPath`. Данный класс позволяет создавать различные контуры из

фигур и текстов. В проекте `Outline` (листинг 2.4) мы рассмотрим, как можно вывести на экран контуры символов заданного текста.

Листинг 2.4. Создание контурного текста

```
'--------------------------------------
' Контур © 2004 Александр Климов
'--------------------------------------
Imports System.Drawing.Drawing2D

    Private Sub Form1_Paint(ByVal sender As Object, ByVal e As
System.Windows.Forms.PaintEventArgs) Handles MyBase.Paint
        ' Создаем траекторию
        Dim pth As New GraphicsPath

        ' Добавляем строку
        pth.AddString("Кошкин дом", _
                      New FontFamily("Tahoma"), _
                      0, 70, New Point(30, 30), _
                      StringFormat.GenericDefault)

        ' Создаем синее перо
        Dim p As New Pen(Color.Blue, 2)

        ' Выводим контурный текст
        e.Graphics.DrawPath(p, pth)

        ' Очистим траекторию
        pth.Reset()

        ' Добавляем новый текст
        pth.AddString("Кошки-мышки", _
                      New FontFamily("Verdana"), _
                      0, 60, New Point(30, 120), _
                      StringFormat.GenericTypographic)

        ' Заливаем траекторию кистью
        e.Graphics.FillPath(Brushes.Peru, pth)
```

```
        ' Выводим на экран
        e.Graphics.DrawPath(p, pth)

        ' Освобождаем ресурсы
        pth.Dispose()
    End Sub
```

Пример достаточно прост для понимания. Создается контур `pth`, в который добавляется первый текст. На экране будут выводиться очертания символов текстовой строки. Для сравнения второй текст выводится, заполненный цветом `Peru` (рис. 2.4). Тем не менее вы все равно можете видеть контур символов.

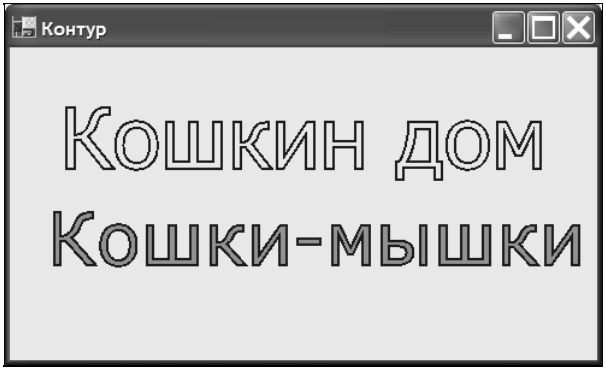

Рис. 2.4. Контуры текста

2.4. Раскаленный текст

Очень неплохо смотрится на экране эффект нагретого до высокой температуры материала или электрического ореола. Способ создания напоминает методику создания объемности. Идею эффекта я почерпнул из примера, расположенного на сайте Роберта Пауэлла (Robert W. Powell) **http://www.bobpowell.net/**. Добавим в проект `ElectricFX` графический объект `PictureBox1`. Местоположение и другие свойства данного элемента несущественны, поэтому оставим все свойства по умолчанию. Весь код содержится в событии `Paint` формы `Form`. В начале кода необходимо добавить ссылку на пространство имен `System.Drawing.Drawing2D` (листинг 2.5).

Листинг 2.5. Создание электрического эффекта

```
'--------------------------------------
' Электрический эффект © 2004 Александр Климов
'--------------------------------------
Imports System.Drawing.Drawing2D

    Private Sub Form1_Paint(ByVal sender As Object, ByVal e As
System.Windows.Forms.PaintEventArgs) Handles MyBase.Paint

        Me.BackColor = Color.Black

        ' Создадим рисунок в заданной пропорции
        Dim bm As New Bitmap(CInt(Me.ClientSize.Width / 5), _
                             CInt(Me.ClientSize.Height / 5))

        ' Создадим объект GraphicsPath
        Dim pth As New GraphicsPath

        ' Добавим строку в заданном стиле
        pth.AddString("Электрический эффект", _
                      New FontFamily("Tahoma"), _
                      CInt(FontStyle.Bold), 48, _
                      New Point(10, 20), _
                      StringFormat.GenericTypographic)

        ' Получим объект Graphics
        Dim g As Graphics = Graphics.FromImage(bm)

        ' Сформируем матрицу для создания эффекта
        Dim mx As Matrix
        mx = New Matrix(1.0F / 5, 0, 0, _
                        1.0F / 5, -(1.0F / 5), _
                       -(1.0F / 5))

        ' Выберем режим сглаживания
        g.SmoothingMode = SmoothingMode.AntiAlias

        ' Преобразуем объект Graphics
        g.Transform = mx
```

```
        ' Создадим перо
        Dim p As New Pen(Color.Tomato, 3)

        ' Рисуем вокруг созданного пути
        g.DrawPath(p, pth)

        ' и заполняем для лучшего эффекта
        g.FillPath(Brushes.Yellow, pth)

        ' Освобождаем ресурсы
        g.Dispose()

        ' Установим режим сглаживания для контура
        e.Graphics.SmoothingMode = SmoothingMode.AntiAlias
        e.Graphics.InterpolationMode = _
                     InterpolationMode.HighQualityBicubic

        ' и расширяем картинку для создания размытости краев
        e.Graphics.DrawImage(bm, ClientRectangle, 0, 0, _
                    bm.Width, bm.Height, GraphicsUnit.Pixel)

        ' Перерисовываем оригинальный текст
        e.Graphics.FillPath(Brushes.Black, pth)

        ' Освобождаем ресурсы
        pth.Dispose()
    End Sub
```

Дадим некоторые пояснения к коду. Чтобы создать ореол вокруг текста, я воспользовался свойством `InterpolationMode`, которое позволяет установить режим размытости текста. Техника создания эффекта ореола основана на двойном наложении текста. Сначала создается уменьшенная копия рисунка, которая будет растянута с применением режима интерполяции. Необходимо установить некоторый коэффициент сжатия картинки. В примере установлен коэффициент 1:5. И вот как это работает. Сначала формируется картинка в заданной пропорции. Затем создается траектория и устанавливается желаемый текст. Теперь можно получить объект `Graphics` из графического объекта `PictureBox` и создать матрицу, которая сжимает картинку. Заполняем траекторию желаемым цветом с помощью пера и для лучшего эффекта заливаем ее еще выбранной кистью. Растягиваем рисунок для получения

эффекта нечеткости (размытости). Снова выводим заданный текст для окончательного создания эффекта (рис. 2.5).

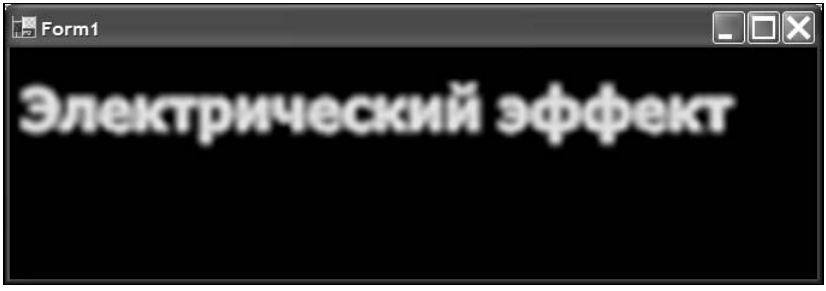

Рис. 2.5. Создание электрического эффекта

2.5. Тексты с поворотами

Особо широкие возможности с текстами предоставляют различные методы класса `Graphics`. Теперь одной строчкой можно повернуть текст на любой угол, отразить зеркально по горизонтали или вертикально, наклонить символы. Аналогичные операции с текстом на Visual Basic 6.0 были доступны только программисту высшей квалификации. Вряд ли в одной книжке можно описать все доступные варианты по трансформации текста. Более подробную информацию вы найдете в документации. Но несколько примеров мы рассмотрим. Я буду приводить статические упражнения, но на их основе вы можете использовать таймер или циклы для создания анимационных эффектов. Вот, например, как можно нарисовать строку, перевернутую на 180 градусов — проект `RotatedText` (листинг 2.6).

Листинг 2.6. Поворот текста на 180 градусов

```
'--------------------------------------
' Поворот текста на 180 градусов © 2004 Александр Климов
'--------------------------------------
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
        Dim g As Graphics
        g = CreateGraphics()

        'Разворачиваем текст на 180 градусов
        g.RotateTransform(180)
```

```
        g.DrawString("Занимательное программирование", _
                     New Font(Font.FontFamily, 16), _
                     Brushes.Green, -Width / 2, -Height / 2)

        g.Dispose()
    End Sub
```

Как видите, для достижения эффекта понадобился всего один вызов метода `RotateTransform` (рис. 2.6).

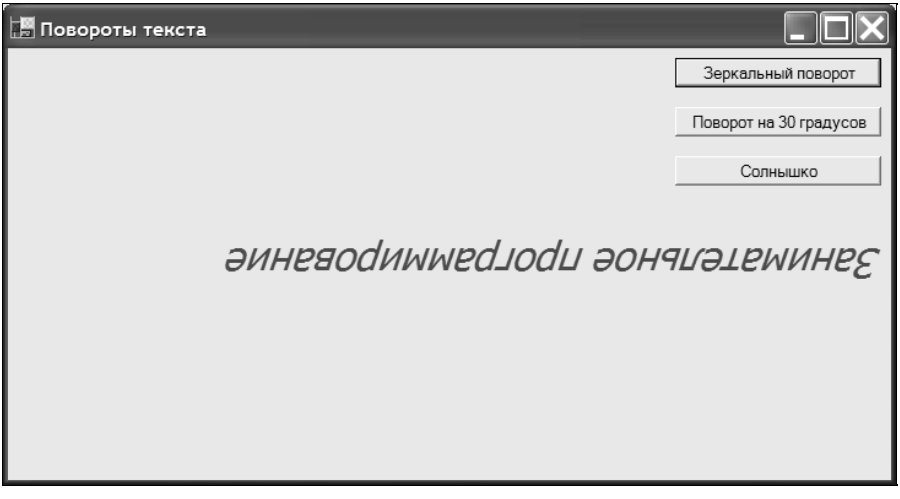

Рис. 2.6. Текст, повернутый на 180 градусов

Соответственно, для небольшого поворота нужно задать требуемое значение угла при помощи метода `RotateTransform` (листинг 2.7).

2.7. Поворот текста на 30 градусов

```
    Private Sub butAngleText_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles butAngleText.Click
        Dim g As Graphics = CreateGraphics()
        g = CreateGraphics()

        ' Перемещаемся в заданную точку отсчета
        g.TranslateTransform(30, 30)
```

```
        'Поворачиваем на 30 градусов
        g.RotateTransform(30)

        ' И выводим текст под заданным углом
        g.DrawString("Текст под углом", f, Brushes.Red, 0, 0)

        'Восстанавливаем преобразование
        g.ResetTransform()
        g.Dispose()
    End Sub
```

Можно использовать не только положительные, но и отрицательные значения. Таким образом, вызов метода `RotateTransform` со значением `-10` начнет поворачивать текст против часовой стрелки. Не могу избавиться от искушения вывести текст вокруг какой-нибудь точки, чтобы получить солнышко (листинг 2.8).

Листинг 2.8. Вывод текста в виде солнышка

```
    Private Sub butSunText_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles butSunText.Click
        g = CreateGraphics()
        ' Очистим форму
        g.Clear(BackColor)

        'Находим центр формы
        Dim cx As Integer = ClientSize.Width \ 2
        Dim cy As Integer = ClientSize.Height \ 2

        ' Перемещаемся в заданную точку отсчета
        g.TranslateTransform(cx, cy)
        g.FillEllipse(Brushes.Yellow, -45, -45, 90, 90)

        Dim i As Integer
        For i = 0 To 359 Step 15
            g.DrawString("Солнышко", f, Brushes.Yellow, 50, 0)
            'Поворачиваем на 20 градусов
            g.RotateTransform(20)
        Next
```

```
        'Восстанавливаем преобразование
        g.ResetTransform()
        g.Dispose()
    End Sub
```

Здесь текст выводится в цикле `For ... Next` с поворотом строки на каждом шаге на двадцать градусов. Чтобы буквы не слипались, я чуть отодвинул вывод текста от центра и добавил желтый кружок для большего реализма (рис. 2.7).

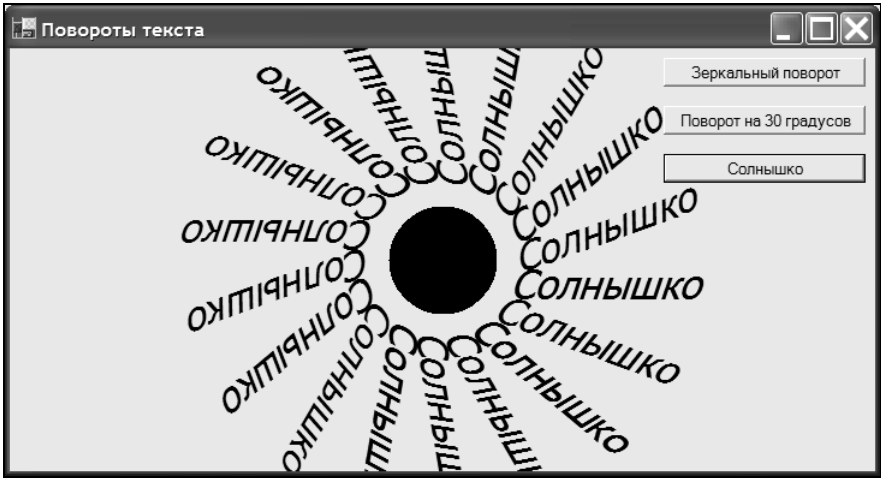

Рис. 2.7. Текст в виде солнышка

2.6. Встречи — расставания

Эффекты, которые я собираюсь вам показать, особенно красиво смотрятся в диалоговых окнах **О программе**, где автор сообщает информацию о себе, версии программы, адрес домашней страницы и электронный адрес. Идея первого примера состоит в следующем. Мы поместим две строчки текста на некотором расстоянии друг от друга. По запуску таймера строки будут сближаться друг с другом, затем немного затормозят при встрече, а потом снова разойдутся. После этого процесс будет повторяться, пока работает таймер. Для этого снова запустим Visual Basic. NET и создадим новый проект под именем `Credits`. Присвоим созданной форме имя `frmAbout`. Добавим на форму также таймер `Timer1`. Все свойства для таймера будут установлены программным путем, поэтому в редакторе свойств ничего менять не будем (листинг 2.9).

Листинг 2.9. Код для примера "Встречи — расставания"

```
'--------------------------------------
' Встречи - расставания © 2004 Александр Климов
'--------------------------------------
    Dim x1 As Single ' координаты для первой надписи
    Dim x2 As Single ' координаты для второй надписи
    Dim g As Graphics
    Dim f As Font
    Dim closuredistance As Single
    Private Const speed As Single = 0.1
    Dim moveamount As Single

    Private Sub Timer1_Tick(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Timer1.Tick
        g.Clear(Me.BackColor)
        g.DrawString("Занимательное", f, Brushes.Blue, x1, 10)
        g.DrawString("Программирование", f, Brushes.BlueViolet, x2, 70)
        closuredistance = Math.Abs(x1 - x2)
        moveamount = speed * closuredistance

        If moveamount < 1 Then moveamount = 1

        x2 = x2 + moveamount
        x1 = x1 - moveamount

        If x2 > Me.Width Or x1 < -1000 Then
            x2 = -270
            x1 = 725
        End If
    End Sub

    Private Sub frmAbout_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        g = Me.CreateGraphics()
        x1 = Me.Width
        x2 = 0
```

```
        f = New Font("Times New Roman", 14, FontStyle.Bold, _
                     GraphicsUnit.Point)
        Timer1.Interval = 40
        Timer1.Start()
    End Sub
```

Код достаточно простой и понятный. В переменную `closuredistance` заносится расстояние между строками с помощью вычитания их координат

```
closuredistance = Math.Abs(x1 - x2)
```

Затем полученное значение умножается на некоторую постоянную величину `speed`. Чем больше расстояние между строками, тем больше значение переменной `moveamount` и, следовательно, выше скорость сближения строк. При сближении скорость перемещения падает. Затем начинается обратный процесс. Надписи набирают скорость и с возрастающим ускорением удаляются друг от друга. Простой и эффектный код, который красиво смотрится на экране.

2.7. Бегущий текст

В *главе 1* я показал, как создать эффект бегущей строки для заголовка формы. В том примере использовались старые функции VB 6.0. Здесь я продемонстрирую технику создания аналогичного примера с применением новых методов класса `String`. Кроме того, в целях придания дополнительной красочности мы будем использовать градиентную заливку текста. Для проекта `RunText` нам понадобится таймер `Timer` (листинг 2.10).

Листинг 2.10. Пример бегущего текста

```
'--------------------------------------
' Бегущая строка © 2004 Александр Климов
'--------------------------------------
    ' Два цвета для градиента
    Private startColor1 As Color = Color.Green
    Private endColor2 As Color = Color.Gold
    ' Шрифт для бегущего текста
    Private f As Font
    ' Для лучшего эффекта нужно добавить
    ' несколько пробелов в конец строки,
    ' чтобы строчки не слипались
    Private sScrollText As String = "Новости           "
```

```
    Public Function RunningText()

        ' Алгоритм эффекта
        ' Удаляем один символ слева
        ' и добавляем его с правой стороны текста
        sScrollText = sScrollText.Substring(1, _
           (sScrollText.Length - 1)) + sScrollText.Substring(0, 1)
        Invalidate()
    End Function

    Private Sub Timer1_Tick(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Timer1.Tick
        RunningText()
    End Sub

    Private Sub Form1_Paint(ByVal sender As Object, ByVal e As
System.Windows.Forms.PaintEventArgs) Handles MyBase.Paint
        ' Создаем градиентную кисть
        Dim MyBrush As Brush = _
            New LinearGradientBrush( _
            ClientRectangle, startColor1, endColor2, 10)
        ' Установим свойства для шрифта
        f = New Font(Font.Name, 60, _
                    Font.Style, GraphicsUnit.Pixel)
        ' Выводим строку
        e.Graphics.DrawString(sScrollText, f, MyBrush, 0, 0)

        ' Освобождаем ресурсы
        MyBrush.Dispose()
        f.Dispose()
    End Sub
```

Запустив проект, вы увидите бегущий справа налево текст. Обратите внимание, что цвет букв во время движения плавно изменяется.

2.8. Скроллинг текста

Но самом деле пример для бегущего текста получился не слишком удачным. Существует гораздо более удобный способ путем изменения значения координат в методе `DrawString`. В проекте `ScrollText` (листинг 2.11) реализова-

на возможность скроллинга текста как в горизонтальном, так и в вертикальном направлении. Для примера нам понадобятся таймер (`tmrScroll`) и два переключателя (`rbVert` и `rbHoriz`).

Листинг 2.11. Скроллинг текста в горизонтальном и вертикальном направлении

```
'--------------------------------------
' Скроллинг текста © 2004 Александр Климов
'--------------------------------------
    ' Для вертикального скроллинга
    Dim y_vert As Single

    ' Для горизонтального скроллинга
    Dim x_horiz As Single

    Dim g As Graphics
    Dim f As Font
    Dim scroll As String = "Скроллинг"

    Private Sub tmrScroll_Tick(ByVal sender As System.Object, _
           ByVal e As System.EventArgs) Handles tmrScroll.Tick

        If rbHoriz.Checked Then
            ' Горизонтальный скроллинг
            g.Clear(Me.BackColor)
            g.DrawString(scroll, f, Brushes.Black, x_horiz, y_vert)
            If x_horiz <= (0 - 90) Then
                x_horiz = Me.Width
            Else
                x_horiz -= 5
            End If
        Else
            ' Вертикальный скроллинг
            g.Clear(Me.BackColor)
            g.DrawString(scroll, f, Brushes.Black, x_horiz, y_vert)

            If y_vert <= (0 - 10) Then
                y_vert = Me.Height
```

```
        Else
            y_vert -= 5
        End If

    End If
End Sub

Private Sub Form1_Load(ByVal sender As System.Object, _
       ByVal e As System.EventArgs) Handles MyBase.Load
    g = Me.CreateGraphics()

    y_vert = Me.Height

    x_horiz = Me.Width - 200

    tmrScroll.Interval = 100
    tmrScroll.Start()
    f = New Font("Times New Roman", 14, FontStyle.Bold, _
       GraphicsUnit.Point)
End Sub
```

Идея скроллинга очень проста. Надо просто вычислить текущие координаты выводимой строки и через интервал, задаваемый таймером, вычитать заданное число пикселов. В коде я использовал фиксированное значение для строки `Скроллинг`:

```
If x_horiz <= (0 - 90) Then ...
```

Значение параметра `90` я подобрал опытным путем. Если вы собираетесь писать универсальный код, подходящий для любой строки, то вам необходимо вычислить длину строки через метод `MeasureString`.

2.9. "Звездные войны"

Видоизменять внешний вид текста, используя мощные средства преобразований, можно самыми различными способами. Например, в некоторых фильмах используется вывод текста с постепенным сужением к верхней части экрана. В частности, такой эффект можно наблюдать в фильме "Звездные войны". Вот простейшая реализация такого приема (листинг 2.12).

Листинг 2.12. Текст в стиле титров фильма "Звездные войны"

```
'--------------------------------------
' Текст в стиле "Звездных войн" © 2004 Александр Климов
'--------------------------------------
Imports System.Drawing.Drawing2D

    Private Sub Form1_Paint(ByVal sender As Object, ByVal e As
System.Windows.Forms.PaintEventArgs) Handles MyBase.Paint
        ' Создаем траекторию
        Dim pth As GraphicsPath = New GraphicsPath
        Dim ff As FontFamily = New FontFamily("Tahoma")

        ' Добавляем в траекторию текст
        Dim y As Integer
        pth.AddString("Жили-были", ff, 0, 70, _
                      New Point(0, 90 * y), _
                      StringFormat.GenericTypographic)

        pth.AddString("старик со старухой", ff, 0, 70, _
                          New Point(0 - 100, 90 * 1), _
                          StringFormat.GenericTypographic)
        pth.AddString("И была у них курочка-ряба", ff, 0, 70, _
                          New Point(0 - 150, 90 * 2), _
                          StringFormat.GenericTypographic)

        ' Создаем массив для преобразования warp
        Dim points() As PointF = { _
New PointF(ClientSize.Width / 2 - ClientSize.Width / 4, 0), _
New PointF(ClientSize.Width / 2 + ClientSize.Width / 4, 0), _
New PointF(0, ClientSize.Height), _
New PointF(ClientSize.Width, ClientSize.Height)}

        ' Преобразуем траекторию
        pth.Warp(points, New RectangleF(0, 0, 820, 450))

        ' Заполняем фон
        e.Graphics.FillRectangle(Brushes.Black, ClientRectangle)
```

```
        ' Заливаем траекторию желтым цветом
        e.Graphics.FillPath(Brushes.Yellow, pth)

        'Освобождаем ресурсы
        pth.Dispose()

    End Sub
```

Если внимательно присмотреться к крайним символам текста (рис. 2.8), то можно заметить, что каждая строчка немного сужается к верху.

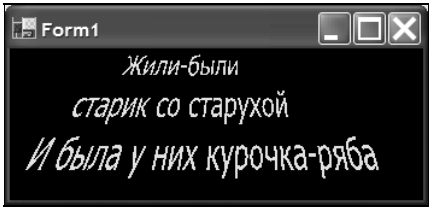

Рис. 2.8. Сужение текста

2.10. Интересные шрифты

Для вывода любого текста на экране используются шрифты, установленные в системе пользователя. Поначалу, мне казалось, что тема шрифтов не интересна и не будет представлена в книге. Но я ошибался. Если подойти к делу творчески, то и здесь можно найти интересные примеры. Кроме обычных шрифтов, которые установлены на большинстве компьютеров, существует масса интересных шрифтов, разрабатываемых как энтузиастами, так и профессиональными фирмами. Возможно, вам захочется использовать такой шрифт в собственных приложениях. Но тут вас подстерегает одна опасность. Если у пользователя не установлен шрифт, используемый вашей программой, то Windows постарается подобрать схожий шрифт. В этом случае вам придется сообщить пользователю о необходимости установить требуемый шрифт. Согласитесь, что это не всегда удобно. Но есть способ лучше. Можно динамически добавлять шрифт в программу. Единственное, что вам придется сделать, так это добавить к программе дополнительный шрифтовый файл. Причем вам даже не понадобится устанавливать свой шрифт в системе (не все пользователи любят засорять системные файлы). К счастью, в Visual Basic .NET 2003 есть класс `PrivateFontCollection`, который и позволяет работать с собственными шрифтами. Если вы внимательно присмотритесь к картинке (рис. 2.9), то увидите, что символы состоят из кошачьих силуэтов! Несколько подобных шрифтов я скачал с сайта **Портал о**

кошках по адресу **http://cats-portal.ru**. Для описываемого ниже примера (проект `CatFont`) я использовал шрифт `PussyFoot` (листинг 2.13). Впрочем, если у вас есть свой интересный шрифт, то можете использовать и его, только не забудьте прописать в коде полный путь к вашему шрифту.

Листинг 2.13. Использование собственных шрифтов в приложениях

```
'--------------------------------------
' Интересные шрифты © 2004 Александр Климов
'--------------------------------------
Imports System.Drawing.Text

    Private Sub Form1_Paint(ByVal sender As Object, ByVal e As
System.Windows.Forms.PaintEventArgs) Handles MyBase.Paint

        Dim count As Integer = 0
        Dim familyName As String = ""
        Dim fontFamilies() As FontFamily
        Dim private_FontColl As New PrivateFontCollection

        ' Добавим свой шрифт в приложение (Используйте свой путь)
        private_FontColl.AddFontFile("e:\download\pussyfoo.ttf")

        fontFamilies = private_FontColl.Families

        ' Вычисляем число семейств шрифта в массиве fontFamilies
        count = fontFamilies.Length

        ' Получим имя первого элемента из коллекции
        familyName = fontFamilies(0).Name

        Dim ff As FontFamily = New FontFamily(familyName, _
                                              private_FontColl)
        Dim f As New Font(ff, 72, FontStyle.Regular)

        ' Выводим строку I LOVE Visual Basic
        e.Graphics.DrawString("I", f, _
                              Brushes.Blue, 150, 20)
        e.Graphics.DrawString("love", f, _
                              Brushes.Red, 70, 120)
```

```
        e.Graphics.DrawString("Visual Basic", f, _
                              Brushes.Blue, 5, 220)
    End Sub
```

Еще раз напоминаю: не забудьте установить свой путь к файлу, содержащему шрифт. Когда я скачал шрифт pussyfoo.ttf, то поместил его в папку e:\download. У вас, вероятно, будет другой путь.

Рис. 2.9. Использование нестандартных шрифтов

Использование нестандартных шрифтов в некоторых случаях вполне оправдано и позволяет придать приложению свою особую привлекательность.

Анекдот в тему

Программист на приеме у глазного врача.

– Вы можете прочитать эту строку таблицы? (показывает "КНШМЫБИ")

– Доктор, у Вас неправильно настроена кодировка.

2.11. Советы и рекомендации

- Для проекта "Скроллинг текста" (*см. разд. 2.8*) переделайте программу так, чтобы текст перемещался сверху вниз и слева направо.
- Используйте в проекте "Скроллинг текста" (*см. разд. 2.8*) вместо фиксированных чисел значения, возвращаемые методом `MeasureString`.

ЧАСТЬ II

Графика

Глава 3

Основы интерфейса GDI+

Пожалуй, самым слабым местом в Visual Basic 6.0 была работа с графикой. Обработка графических объектов большого размера могла растянуться на десятки минут. А то дело могло закончиться и зависанием компьютера. Кроме того, встроенных возможностей Visual Basic хватало только на самые простейшие операции. Для более сложных приемов работы с графикой приходилось изучать сложные функции Windows API. Теперь все изменилось. Работа с графикой доставляет сплошное удовольствие. Мы рассмотрим наиболее интересные примеры.

Интерфейс GDI+ является следующим этапом развития старого интерфейса GDI, о чем свидетельствует знак плюса в названии. Сразу хочу оговориться, что, несмотря на весьма широкие возможности, предоставляемые новым интерфейсом, в отдельных случаях нам еще придется обращаться к неуправляемому коду через функции Windows API. Это вполне естественно, так как разработчики .NET Framework просто не в состоянии за короткие сроки реализовать все необходимые графические методы и свойства, в которых так нуждается программист. На данный момент в состав GDI+ входят шесть пространств имен:

- `System.Drawing`;
- `System.Drawing.Design`;
- `System.Drawing.Drawing2D`;
- `System.Drawing.Imaging`;
- `System.Drawing.Printing`;
- `System.Drawing.Text`.

Для нас, пожалуй, наибольший интерес представляют первые четыре пространства имен, связанные непосредственно с графикой и изображениями. Для программистов, которые очень плотно работали с предыдущим интерфейсом GDI, будет интересно узнать основные различия между старой и новой технологией. Прежде всего больше не используются понятия контекста устройства (`Device Context`) и дескриптора контекста устройства. На смену этим понятиям пришел объект `Graphics`, в котором и содержатся не-

обходимые для рисования методы и свойства. Кроме того, изменились принципы вывода графики на устройства. Например, раньше можно было вызвать функцию создания пера и рисовать линии этим пером неограниченное число раз. Теперь надо явно указывать используемое перо при каждом выводе линии на экране. Уже этих двух примеров достаточно, чтобы понять, что графические приемы претерпели значительные изменения. Так что будьте предельно внимательны при изучении данной главы.

3.1. Класс *Graphics*

Класс `Graphics` — один из самых мощных и обширных классов в иерархии .NET Framework, обеспечивающий работу с графикой. С его помощью можно рисовать линии, кривые, фигуры. Одно только перечисление свойств и методов класса займет несколько страниц текста. Я выбрал только самые интересные на мой взгляд примеры. Но сначала надо немного изучить теорию и познакомиться с основными объектами графики. Самое первое, что нужно запомнить — прежде чем рисовать какой-либо объект (квадрат, эллипс, линию), — надо создать поверхность для рисования (то, что раньше называлось *контекстом устройства*). Поверхностью может служить и сама форма, и весь экран, и элемент управления, а также принтер. Для создания такой поверхности служит объект `Graphics`. Получить доступ или создать данный объект можно разными способами:

- во-первых, сама форма уже имеет объект `Graphics`, к которому можно обращаться через события, связанные с рисованием (например, `Paint`);
- если же нужно обратиться к данному объекту через события кнопки, то можно воспользоваться методом `CreateGraphics`.

Это основные приемы, которыми я буду пользоваться в данной книге. Другие способы применяются реже и рассматриваться не будут. Хочу особо остановиться на методе `CreateGraphics`. Новички зачастую совершают одну и ту же ошибку при вызове данного метода:

```
Dim g As New Graphics
```

Данный код работать не будет, и редактор кода укажет на ошибку. Вот правильный вариант:

```
Dim g As Graphics
```

В этой книге я часто использую следующий прием:

```
Dim g As Graphics = CreateGraphics()
```

Конечно, если вас смущает всего одна буква `g` для определения целого объекта, то можете использовать и другие варианты: `gr`, `grfx` и любой другой. Но данный объект так часто используется в моих примерах, что мне проще набирать на клавиатуре одну букву, экономя свое драгоценное время.

3.2. Перья

После создания поверхности для рисования можно начать рисовать. Как и в реальной жизни, рисовать можно пером, кистью и целыми картинками. Давайте сравним эти способы с нашими представлениями о рисовании. Пером или карандашом (как более современный и понятный вариант) удобно рисовать линии, кривые, дуги. В принципе, карандашом можно и закрасить нарисованную окружность, но это уже достаточно утомительно и не производительно. Карандаш может рисовать толстые и тонкие линии в зависимости от силы нажатия на карандаш, его твердости и степени заточки. Карандаш также может быть цветным. На языке машины это выглядит следующим образом:

```
Dim p As New Pen(Color.Red)
Dim p2 As New Pen(Color.Blue, 5) ' жирный карандаш
```

В первом варианте создается перо красного цвета в один пиксел. Во втором случае уже используется более широкое перо синего цвета в пять пикселов.

3.2.1. Метод *DrawLine*

Метод `DrawLine` позволяет рисовать линии разного цвета и толщины. При создании линий используются перо и координаты двух крайних точек. Так как цвет пера можно создавать на основе метода `FromArgb`, то у нас есть возможность создавать линии с разным уровнем прозрачности — проект `DrawLineSample` (листинг 3.1).

Листинг 3.1. Рисование линий

```
'--------------------------------------
' Полупрозрачные линии © 2004 Александр Климов
'--------------------------------------
    Private Sub Form1_Paint(ByVal sender As Object, ByVal e As
System.Windows.Forms.PaintEventArgs) Handles MyBase.Paint
        Dim g As Graphics = e.Graphics
        Dim clr As New Color
        Dim p As New Pen(clr, 5)

        ' выводим ряд горизонтальных линий
        ' с изменяющейся прозрачностью
        Dim i As Integer
        For i = 10 To 210 Step 20
            p.Color = clr.FromArgb(255 - i, i, 255, 0)
```

```
            g.DrawLine(p, 10, i, 200, i)
        Next
    End Sub
```

Запустив проект, вы увидите ряд горизонтальных линий с изменяющейся прозрачностью. Для усиления эффекта можно добавить перед этим кодом еще несколько линий со сплошным цветом:

```
g.DrawLine(New Pen((Color.Red), 10), 20, 5, 20, 215)
g.DrawLine(New Pen((Color.Blue), 10), 190, 5, 190, 215)
```

3.3. Кисти

Кистью удобно закрашивать большие пространства. Здесь более уместна ассоциация с малярной кистью, чем с кистью художника. В отличие от класса `Pen`, класс `Brush` является абстрактным, и создать экземпляр класса на его основе нельзя. Я в первое время постоянно об этом забывал и писал:

```
Dim br as New Brush(Color.Black)
```

А потом долго удивлялся, где же я совершил ошибку? Не повторяйте моих ошибок. Пишите правильно:

```
Dim sb as New SolidBrush(Color.Black)
```

Существует несколько типов кистей — простых и продвинутых, которые находятся в пространствах имен `System.Drawing` и `System.Drawing.Drawing2D`. Кисти являются важной частью графической системы Windows. С точки зрения занимательности, пожалуй, представляют для нас интерес классы `TextureBrush`, `LinearGradientBrush` и `PathGradientBrush`. Хотя, безусловно, и другие классы типа `SolidBrush` и `HatchBrush` тоже могут стать источником интересных примеров. Напоминаю, что все эти классы являются производными от главного абстрактного класса `Brush`.

3.3.1. Класс *SolidBrush*

Прежде чем я познакомлю вас с новыми кистями, давайте сначала вспомним, как использовать сплошную кисть в проектах (проект `SolidBrushSample`, листинг 3.2).

Листинг 3.2. Использование сплошных кистей

```
'--------------------------------------
' Сплошные кисти © 2004 Александр Климов
'--------------------------------------
```

```
   Private Sub butSimple_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles butSimple.Click
        Dim g As Graphics = CreateGraphics()
        Dim rect As New Rectangle(10, 10, _
                Me.ClientSize.Width - 20, _
                Me.ClientSize.Height - 20)
        ' Создаем зеленую сплошную кисть
        Dim sbr As Brush = New System.Drawing.SolidBrush(Color.Green)

        ' Заполняем прямоугольник зеленой кистью.
        g.FillEllipse(sbr, rect)

        ' Обрамляем прямоугольник эллипсом в 10 пикселов.
        g.DrawEllipse( _
            New Pen(Color.DarkMagenta, 10), rect)
        g.Dispose()
    End Sub
```

Но сплошная кисть может быть не только однородной, но и полупрозрачной. Вот как можно создать три заполненные полупрозрачные окружности разного цвета (листинг 3.3).

Листинг 3.3. Использование полупрозрачных кистей

```
    Private Sub butTransp_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles butTransp.Click
        Dim g As Graphics = CreateGraphics()
        ' Создадим три полупрозначные кисти.
        Dim trnsRedBrush As New SolidBrush(_
                        Color.FromArgb(150, 255, 0, 0))
        Dim trnsGreenBrush As New SolidBrush (_
                        Color.FromArgb(150, 0, 255, 0))
        Dim trnsBlueBrush As New SolidBrush (_
                        Color.FromArgb(150, 0, 0, 255))
        ' Основание и высота треугольника
        ' для позиционирования кругов
        ' Каждая вершина треугольника является центром
        ' одного из кругов.
        ' Основание равно диаметру круга.
```

```
    Dim triBase As Single = 150
    Dim triHeight As Single = CSng(Math.Sqrt((3 * (triBase * _
    triBase) / 4)))

    ' Координаты для первого круга.
    Dim x1 As Single = 40
    Dim y1 As Single = 40

    ' Заполняем круги.
    g.FillEllipse(trnsRedBrush, x1, y1 _
                  2 * triHeight, 2 * triHeight)
    g.FillEllipse(trnsGreenBrush, x1 + triBase / 2 _
                  y1 + triHeight, 2 * triHeight, 2 * triHeight)
    g.FillEllipse(trnsBlueBrush, x1 + triBase, _
                  y1, 2 * triHeight, 2 * triHeight)

    g.Dispose()
End Sub
```

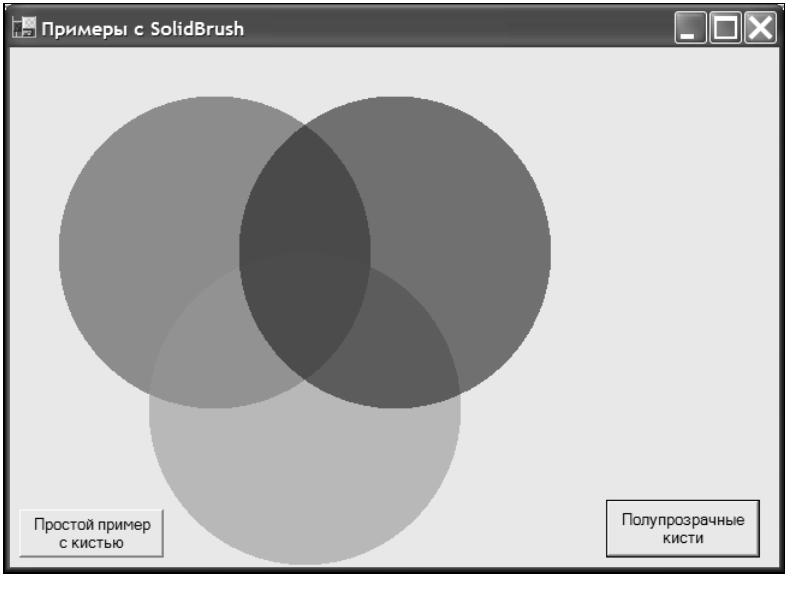

Рис. 3.1. Окружности с использованием полупрозрачных кистей

Попробуйте запускать проект в разных режимах. Например, сначала нажмите на кнопку с надписью **Полупрозрачные кисти**. Вы увидите описанный только что пример (рис. 3.1). Вашему взору предстанут три полупрозрачных круга, частично перекрывающие друг друга. Посмотрите на получаемые цвета в области пересечения кругов. Теперь попробуйте нажать на кнопку с надписью **Простой пример с кистью**, а затем снова нажать на вторую кнопку. Обратите внимание, что на зеленом фоне большого эллипса полупрозрачный красный круг получается коричневым.

3.3.2. Класс *HatchBrush*

Класс `HatchBrush` содержит более 50 различных узорных кистей. Для сравнения — в Windows GDI содержится всего шесть стилей. Почувствуйте разницу! Для наглядности попробуем отобразить все имеющиеся стили в проекте `HatchBrushStyles` (листинг 3.4).

Листинг 3.4. Отображение всех стилей класса `HatchBrush`

```
'--------------------------------------
' Все стили класса HatchBrush © 2004 Александр Климов
'--------------------------------------
Imports System.Drawing.Drawing2D


    Private Sub Form1_Paint(ByVal sender As Object, ByVal e As
System.Windows.Forms.PaintEventArgs) Handles MyBase.Paint
        ' Всего имеется 53 стиля
        Const FIRST_STYLE As Integer = 0
        Const LAST_STYLE As Integer = 52
        Dim x As Integer = 10
        Dim y As Integer = 10

        Dim i As Integer
        For i = FIRST_STYLE To LAST_STYLE
            DrawAllHatch(e.Graphics, x, y, CType(i, HatchStyle))
        Next i
    End Sub


    Private Sub DrawAllHatch(ByVal g As Graphics, ByRef x As Integer,
ByRef y As Integer, ByVal hatch_style As HatchStyle)
        ' Ширина текста
        Const TEXT_WID As Integer = 170
```

```
        ' Размеры прямоугольников, содержащих образец кисти
        Dim rect As New Rectangle(x + TEXT_WID, y, 40, 35)

        ' Используем самую простую кисть
        Dim hbr As New HatchBrush(hatch_style, Color.White)
        g.DrawString(CInt(hatch_style) & ". " & _
            hatch_style.ToString, _
            Me.Font, Brushes.Black, x, y)
        g.FillRectangle(hbr, rect)
        g.DrawRectangle(Pens.Black, x + TEXT_WID, y, rect.Width,
rect.Height)

        y += rect.Height + 10
        If y + rect.Height > Me.ClientSize.Height Then
            x += TEXT_WID + rect.Width + 30
            y = 10
        End If
    End Sub
```

Данный пример выводит все стили в небольших прямоугольных областях (рис. 3.2). Причем, обратите внимание, что каждый прямоугольник пронумерован и снабжен названием стиля. Лично мне нравится стиль `HorizontalBrick`, который состоит из множества кирпичиков. Не забывайте, что кисть может иметь не только черно-белую составляющую, но и любую пару цветов на ваш выбор. Давайте закрасим форму в виде участка Кремлевской стены красными кирпичами (листинг 3.5).

Листинг 3.5. Заполнение формы кирпичами

```
    Private Sub butBricks_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles butBricks.Click
        ' Заполним всю форму кирпичами
        Dim hbr As New HatchBrush(_
                  HatchStyle.HorizontalBrick, Color.Gray, Color.Tomato)
        Dim g As Graphics = CreateGraphics()

        g.FillRectangle(hbr, 0, 0, ClientSize.Width, ClientSize.Height)
        g.Dispose()
    End Sub
```

Обратите внимание, что красный цвет считается задним фоном, а передним фоном являются швы между кирпичами. Я поначалу путался с этими значениями.

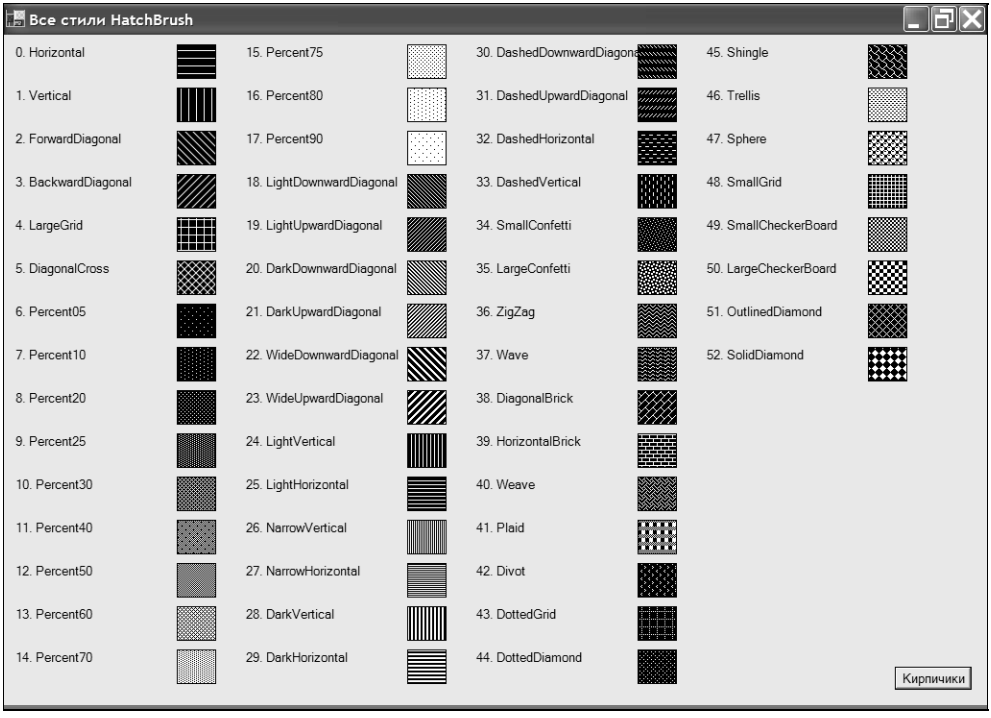

Рис. 3.2. Все стили узорной кисти

3.3.3. Класс *TextureBrush*

Особенностью текстурных кистей, созданных на основе класса `TextureBrush`, является использование картинки. Приведем сначала самый простой пример применения текстурной кисти — проект `TextureBrushSample` (листинг 3.6).

Листинг 3.6. Использование текстурных кистей

```
'--------------------------------------
' Текстурные кисти © 2004 Александр Климов
'--------------------------------------

    Private Sub butSimpleTextureBrush_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
butSimpleTextureBrush.Click
```

```
        Dim g As Graphics
        g = CreateGraphics()
        Dim img As Image = PictureBox1.Image
        Dim tb As New System.Drawing.TextureBrush(img _
                              New Rectangle(0, 0, 60, 88))

        tb.WrapMode = Drawing2D.WrapMode.Tile.TileFlipY
        g.FillEllipse(tb, 0, 0, _
                   2 * ClientSize.Width \ 3, 2 * ClientSize.Height \ 3)
        g.FillRectangle(tb, ClientSize.Width \ 3, _
                            ClientSize.Height \ 3, _
                            ClientSize.Width \ 2, _
                            2 * ClientSize.Height \ 3)
        g.Dispose()
    End Sub
```

В этом примере мы создаем текстурную кисть, где в качестве картинки служит предварительно загруженное изображение из графического объекта `PictureBox`. У текстурных кистей существует несколько режимов заполнения. Например, режим `TileFlipY` позволяет отразить картинку по горизонтали и размножить ее, заполняя эллипс и прямоугольник. Обратите внимание, что, несмотря на наложение этих двух фигур, рисунки в них совпадают, не перекрывая друг друга. К слову сказать, текстурные кисти можно вращать, сдвигать, масштабировать. Поставьте соответствующий вызов перед выводом фигур на экран:

```
tb.RotateTransform(-35)
```

Как видите, узор кистей повернулся на тридцать пять градусов.

Можно заполнять кистью не только готовые фигуры, вроде прямоугольника и эллипса, но и собственную фигуру, созданную с помощью массива точек (рис. 3.3). Для этого можно воспользоваться методом `DrawPolygon` (листинг 3.7).

Листинг 3.7. Заполнение кистью сложной фигуры

```
    Private Sub butPolygon_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles butPolygon.Click
        Dim pts() As Point = { _
            New Point(50, 25), _
            New Point(100, 25), _
            New Point(150, 100), _
            New Point(100, 175), _
```

```
            New Point(50, 175), _
            New Point(5, 100) _
        }

        ' Заполняем полигон кистью.
        Dim tb As New System.Drawing.TextureBrush(PictureBox1.Image)
        Dim g As Graphics
        g = CreateGraphics()
        g.Clear(Me.BackColor)
        g.FillPolygon(tb, pts)

        ' Создаем перо.
        Dim a_pen As New Pen(Color.RoyalBlue, 2)
        a_pen.DashStyle = Drawing.Drawing2D.DashStyle.DashDotDot

        ' Рисуем полигон.
        g.DrawPolygon(a_pen, pts)
        g.Dispose()
    End Sub
```

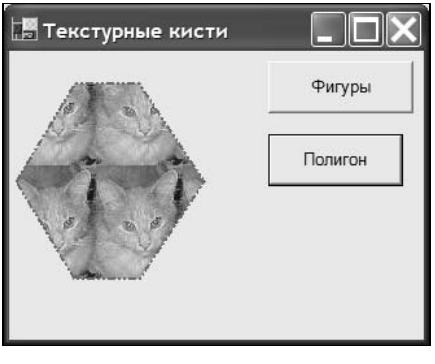

Рис. 3.3. Использование текстурных кистей

Приведенный пример выглядит слишком статичным. Поэтому предлагаю рассмотреть другой вариант применения текстурных кистей. На этот раз мы будем рисовать на форме различные фигуры, которые автоматически заполнятся выбранной кистью.

Основные действия будут находиться в событиях `MouseDown`, `MouseMove` и `MouseUp`. Для начала определим несколько глобальных переменных в новом проекте `TextureBrush2` (листинг 3.8).

Листинг 3.8. Динамическое заполнение фигуры кистью

```
'--------------------------------------
' Текстурные кисти-2 © 2004 Александр Климов
'--------------------------------------

    Private AllPoints() As Point
    Private MaxPoint As Integer
    Private bDrawing As Boolean

    Private Sub Form1_MouseDown(ByVal sender As Object, ByVal e As
System.Windows.Forms.MouseEventArgs) Handles MyBase.MouseDown
        bDrawing = True
        Dim g As Graphics = Me.CreateGraphics()
        g.Clear(Me.BackColor)

        MaxPoint = 0
        ReDim AllPoints(MaxPoint)
        AllPoints(MaxPoint).X = e.X
        AllPoints(MaxPoint).Y = e.Y
    End Sub

    Private Sub Form1_MouseMove(ByVal sender As Object, ByVal e As
System.Windows.Forms.MouseEventArgs) Handles MyBase.MouseMove
        If Not bDrawing Then Exit Sub
        MaxPoint += 1
        ReDim Preserve AllPoints(MaxPoint)
        AllPoints(MaxPoint).X = e.X
        AllPoints(MaxPoint).Y = e.Y

        Dim g As Graphics = Me.CreateGraphics()
        g.DrawLine(Pens.Black, _
            AllPoints(MaxPoint - 1), AllPoints(MaxPoint))
    End Sub

    Private Sub Form1_MouseUp(ByVal sender As Object, ByVal e As
System.Windows.Forms.MouseEventArgs) Handles MyBase.MouseUp
        bDrawing = False
        If MaxPoint < 1 Then Exit Sub
```

```
        Dim g As Graphics = Me.CreateGraphics()
        Dim br As New TextureBrush(PictureBox1.Image)
        g.FillPolygon(br, AllPoints)
        g.DrawPolygon(Pens.Black, AllPoints)
        g.Dispose()
    End Sub
```

Разберем код по порядку. Когда пользователь начинает рисовать, то он сначала нажимает на кнопку мыши, тем самым активируя событие `MouseDown`. В обработчике данного события используется флаг `bDrawing`, который показывает, что рисование разрешено. Затем содержимое формы очищается с помощью метода `Clear`, и в массив `AllPoints` помещаются координаты мыши. Нажав кнопку, пользователь начинает перемещать мышь. Именно в этом состоянии происходит собственно рисование. Следовательно, требуется обработка события `MouseMove`. При перемещении мыши мы записываем в массив изменяющие координаты мыши и соединяем их с помощью метода `DrawLine`. Осталось только залить получившуюся фигуру выбранной текстурной кистью, что мы и делаем при отпускании кнопки мыши (событие `MouseUp`). Запустите пример и нарисуйте любую фигуру в любом направлении. Как только закончите рисовать (отпустите кнопку мыши), картинка заполнится текстурной кистью, основой которой послужило нам изображение, предварительно загруженное нами в графический объект `PictureBox` (рис. 3.4).

Рис. 3.4. Эффекты с кистями

3.4. Прямоугольники

Для рисования одного прямоугольника используются методы `DrawRectangle` или `FillRectange`, для рисования нескольких прямоугольников удобнее

применить методы `DrawRectangles` или `FillRectangles`. При использовании методов с префиксом `Fill` заданные прямоугольники заливаются кистью заданного цвета. В остальных случаях рисуется окантовка фигуры. Вот небольшой пример вывода последовательности синих квадратов по диагонали (квадрат тоже является прямоугольником, если вдруг вы забыли) в проекте `Figures` (листинг 3.9).

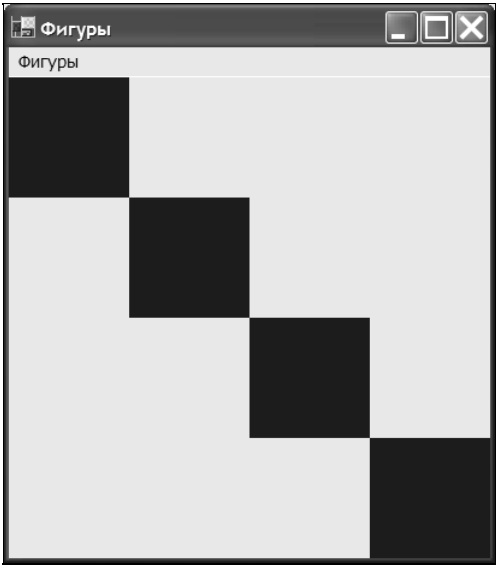

Рис. 3.5. Рисование квадратов

Листинг 3.9. Рисование прямоугольников

```
'--------------------------------------
' Прямоугольники © 2004 Александр Климов
'--------------------------------------

    Private Sub Form1_Load(ByVal sender As Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        ClientSize = New Size(400, 400)
    End Sub


    Private Sub mnuDrawRectangles_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles mnuDrawRectangles.Click
        ' Выводим последовательность квадратов по диагонали
        Dim g As Graphics = CreateGraphics()
```

```
        Dim rs As Rectangle() = {New Rectangle(0, 0, 100, 100), _
                                 New Rectangle(100, 100, 100, 100), _
                                 New Rectangle(200, 200, 100, 100), _
                                 New Rectangle(300, 300, 100, 100)}
        g.FillRectangles(New SolidBrush(Color.Blue), rs)
        g.Dispose()
    End Sub
```

Я специально задал размеры клиентской области формы равным 400 × 400 пикселов, чтобы точно подогнать квадраты под ее размеры (рис. 3.5).

3.5. Эллипсы

Теперь, когда вы слегка познакомились с перьями и кистями, можно приступать к рисованию других простых фигур. Для рисования эллипсов используются методы `DrawEllipse` или `FillEllipse`. Метода, рисующего последовательность эллипсов аналогично методу `DrawRectangles`, в VB .NET 2003 не существует. Но это не трудно реализовать самому. В следующем примере (листинг 3.10) рисуются несколько вложенных эллипсов (рис. 3.6).

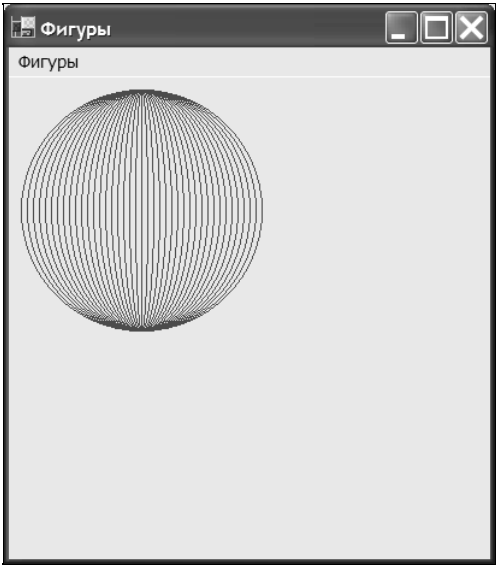

Рис. 3.6. Рисование вложенных эллипсов

Листинг 3.10. Рисование эллипсов

```
    Private Sub mnuEllipse_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles mnuEllipse.Click
        Dim g As Graphics = CreateGraphics()
        Dim p As New Pen(Color.Green)
        Dim i As Integer
        For i = 0 To 100 Step 5
            g.DrawEllipse(p, 10 + i, 10, 200 - 2 * i, 200)
        Next
        g.Dispose()
    End Sub
```

3.6. Секторы

Если вы любите есть пироги, то вам знакома фигура сектора — кусочек разрезанного пирога как раз ей и соответствует. А рисуется такой кусочек с помощью метода `FillPie`. Возможно, вам приходилось видеть знак, рисуемый на таре, содержащей радиоактивные вещества (рис. 3.7). Вот приблизительно как это реализуется программным способом (листинг 3.11).

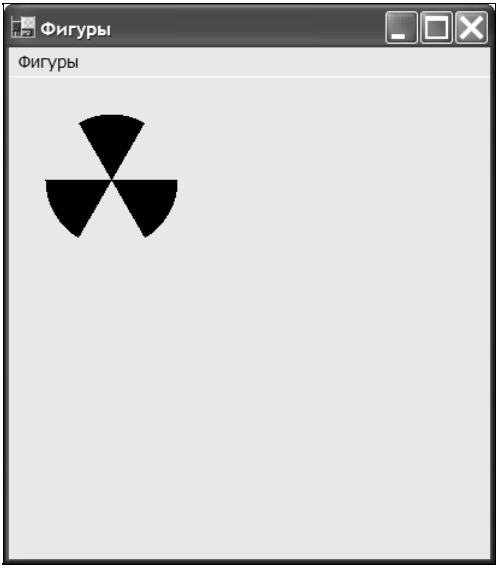

Рис. 3.7. Рисование секторов

Листинг 3.11. Рисование секторов

```
    Private Sub mnuPie_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles mnuPie.Click
        Dim g As Graphics = CreateGraphics()
        Dim sb As New SolidBrush(Color.Yellow)
        Dim i As Integer
        For i = 0 To 359 Step 120
            g.FillPie(sb, 30, 30, 110, 110, i, 60)
        Next
        g.Dispose()
    End Sub
```

3.7. Дуги

Для рисования дуг используется метод `DrawArc`. С помощью двойного вызова метода можно нарисовать иллюстрацию к песне "Светит месяц, светит ясный" (листинг 3.12).

Листинг 3.12. Рисование дуг

```
    Private Sub mnuArc_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles mnuArc.Click
        Dim g As Graphics = CreateGraphics()
        g.DrawArc(New Pen(Color.Red), 100, 70, 300, 300, 90, -180)
        g.DrawArc(New Pen(Color.Red), 150, 70, 200, 300, 90, -180)
        g.Dispose()
    End Sub
```

3.8. Многоугольники

Многоугольники задаются массивом точек, которые соединяются между собой прямыми отрезками с помощью метода `DrawPolygon`, причем первая точка массива автоматически соединяется с последней его точкой. В самом методе нет ничего интересного и сложного для нас, поэтому рассмотрим пример динамического создания многоугольника. Для этой цели нужно создать массив из 10 точек, координаты которых можно будет менять перетаскиванием вершин при помощи указателя мыши (листинг 3.13).

Листинг 3.13. Создание многоугольника

```
'---------------------------------
' Многоугольник © 2004 Александр Климов
'---------------------------------
Imports System.Math

    ' Переменная, устанавливающая состояние перемещения
    ' Если равна -1, то перемещать нельзя
    Private CheckState As Integer = -1

    ' Объявляем массив точек
    Private apts(9) As Point

    ' Размеры меток
    Private Const iSize As Integer = 6
    Private Const iCenter As Integer = iSize \ 2

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As Sys-
tem.EventArgs) Handles MyBase.Load

        ' Задаем вершины многоугольника
        apts(0).X = 80 : apts(0).Y = 20
        apts(1).X = 100 : apts(1).Y = 80
        apts(2).X = 160 : apts(2).Y = 80
        apts(3).X = 120 : apts(3).Y = 120
        apts(4).X = 140 : apts(4).Y = 180
        apts(5).X = 80 : apts(5).Y = 140
        apts(6).X = 20 : apts(6).Y = 180
        apts(7).X = 40 : apts(7).Y = 120
        apts(8).X = 5 : apts(8).Y = 80
        apts(9).X = 60 : apts(9).Y = 80
    End Sub

    Private Sub CreatePolygon(ByVal g As Graphics)
        'Очистим формы от предыдущих линий
        g.Clear(BackColor)
        ' Рисуем многоугольник
        g.DrawPolygon(Pens.Red, apts)
```

```
        ' Рисуем метки, за которые можно ухватиться мышью
        Dim rectangles() As Rectangle
        ReDim rectangles(apts.GetUpperBound(0))
        For i As Integer = 0 To apts.GetUpperBound(0)
            'Определяем размеры меток и их центры
            With rectangles(i)
                .X = apts(i).X - iCenter
                .Y = apts(i).Y - iCenter
                .Width = iSize
                .Height = iSize
            End With
        Next i

        ' Заполняем метки белым цветом и обрамляем их черным цветом
        g.FillRectangles(Brushes.White, rectangles)
        g.DrawRectangles(Pens.Black, rectangles)
    End Sub

    Private Sub Form1_Paint(ByVal sender As Object, ByVal e As
System.Windows.Forms.PaintEventArgs) Handles MyBase.Paint
        CreatePolygon(e.Graphics)
    End Sub

    Private Sub Form1_MouseDown(ByVal sender As Object, ByVal e As
System.Windows.Forms.MouseEventArgs) Handles MyBase.MouseDown
        Dim dx As Single
        Dim dy As Single

        For i As Integer = 0 To apts.GetUpperBound(0)
            If Abs(apts(i).X - e.X) < iCenter And _
               Abs(apts(i).Y - e.Y) < iCenter _
            Then
                ' Начинаем тащить
                CheckState = i
                Exit For
            End If
        Next i
    End Sub
```

```
    Private Sub Form1_MouseMove(ByVal sender As Object, ByVal e As
System.Windows.Forms.MouseEventArgs) Handles MyBase.MouseMove
        ' Ничего не делаем, если нет перемещения
        If CheckState = -1 Then Exit Sub

        ' Перемещаем вершины
        apts(CheckState).X = e.X
        apts(CheckState).Y = e.Y

        ' Перерисовываем
        CreatePolygon(Me.CreateGraphics())
    End Sub

    Private Sub Form1_MouseUp(ByVal sender As Object, ByVal e As
System.Windows.Forms.MouseEventArgs) Handles MyBase.MouseUp
        CheckState = -1
    End Sub
```

Запустив программу, вы увидите многоугольник в виде звезды, вершины которой имеют специально созданные маленькие квадратики. Если подвести мышь к одному из этих квадратиков и, нажав левую кнопку, потащить его в сторону, то вид многоугольника изменится. У нас получился своеобразный редактор многоугольников (рис. 3.8).

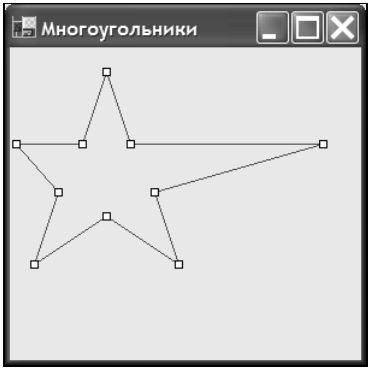

Рис. 3.8. Создание многоугольников

3.9. Кривые Безье

Кривые Безье являются особым видом кривых, поведение которых зависит от четырех заданных точек. Разработаны они были французским ученым

Пьером Безье для нужд автомобилестроения. В наше время кривые Безье широко применяются и в других областях науки и промышленности. Но нас они должны интересовать с эстетической точки зрения. Рисование кривых Безье является достаточно интересным развлечением. Поведение кривых Безье можно представить следующим образом — кривая выходит из первой, заданной, точки и заканчивается в последней, четвертой точке. Две промежуточные точки являются своеобразными магнитами. Кривая сначала пытается приблизиться к первой промежуточной точке, но, не доходя до нее, устремляется ко второй промежуточной точке, уворачивается и от нее и заканчивается в последней точке. За поведением такой кривой интересно наблюдать в интерактивном режиме. Создадим модель (проект `Beziers`), в которой мы с вами будем устанавливать четыре точки в произвольных местах, и кривая Безье станет автоматически рисоваться по нашим установленным точкам (листинг 3.14).

Листинг 3.14. Рисование кривой Безье по заданным точкам

```
'--------------------------------------
' Кривые Безье © 2004 Александр Климов
'--------------------------------------
    ' Для кривой Безье необходимо иметь четыре точки
    Private m_Points(3) As Point
    Private LimitPoints As Integer = -1

    Private Sub Form1_MouseDown(ByVal sender As Object, ByVal e As
System.Windows.Forms.MouseEventArgs) Handles MyBase.MouseDown
        Dim g As Graphics = CreateGraphics()

        If LimitPoints = 3 Then
            g.Clear(Me.BackColor)
            LimitPoints = 0
        Else
            LimitPoints += 1
        End If

        m_Points(LimitPoints).X = e.X
        m_Points(LimitPoints).Y = e.Y

        ' Рисуем четыре окружности
        For i As Integer = 0 To LimitPoints
            Dim rect As New Rectangle( _
```

```
                m_Points(i).X - 5, m_Points(i).Y - 5, 10, 10)
            g.FillEllipse(Brushes.Blue, rect)
        Next i

        If LimitPoints = 3 Then
            ' Выводим кривую Безье
            g.DrawBezier(Pens.Red, _
                m_Points(0), m_Points(1), _
                m_Points(2), m_Points(3))
        End If

        g.Dispose()

    End Sub
```

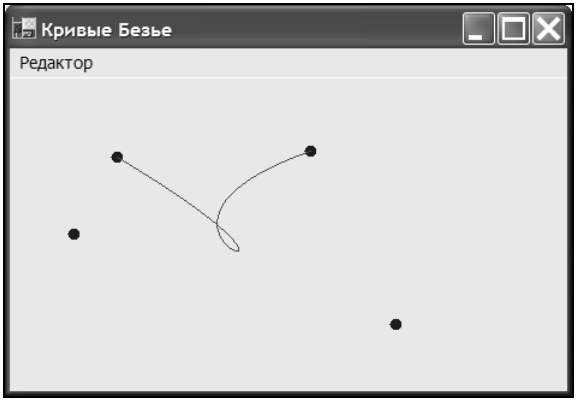

Рис. 3.9. Рисование кривой Безье по заданным точкам

Запустив программу, щелкайте в произвольных местах формы. После четвертого щелчка на форме появится кривая Безье, проходящая в соответствии с вашими контрольными точками. Чтобы процесс был более нагляден, точки в примере заменены на синие кружочки (рис. 3.9).

Пример получился неплохой, но какое-то чувство неудовлетворенности все равно осталось. Хочется иногда просто подвигать за контрольные точки, чтобы получить более выразительный пример кривой. Я решил реализовать небольшой редактор кривых Безье. Смысл редактирования заключается в следующем. На форму выводится некая кривая Безье. При щелчке левой кнопкой мыши курсор устанавливается в начальную точку кривой. Теперь, не отпуская кнопки мыши, переместите начальную точку в другое место

формы. При щелчке правой кнопкой мыши курсор, соответственно, устанавливается в конечную точку кривой. Аналогично, ее также можно перемещать по форме. При перемещении мыши меняется и вид кривой Безье. Для экономии места я решил не создавать новый проект, а дополнить предыдущий пример (листинг 3.15).

Листинг 3.15. Редактор кривых Безье

```
    Dim startpoint As New Point(100, 100)
    Dim control1 As New Point(200, 10)
    Dim control2 As New Point(350, 20)
    Dim endpoint As New Point(400, 100)

    Private Sub Form1_MouseDown(ByVal sender As Object, _
           ByVal e As System.Windows.Forms.MouseEventArgs) _
           Handles MyBase.MouseDown

            ' При нажатии на левую кнопку мыши
            If e.Button = MouseButtons.Left Then
                Cursor.Position = PointToScreen(startpoint)
            ' При нажатии на правую кнопку мыши
            ElseIf e.Button = MouseButtons.Right Then
                Cursor.Position = PointToScreen(endpoint)
            End If
    End Sub

    Private Sub mnuEditBezier_Click(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles mnuEditBezier.Click
        Dim g As Graphics = CreateGraphics()
        g.Clear(Me.BackColor)
        g.DrawBezier(Pens.Red, startpoint, control1, control2, endpoint)
    End Sub

    Private Sub Form1_MouseMove(ByVal sender As Object, _
           ByVal e As System.Windows.Forms.MouseEventArgs) _
           Handles MyBase.MouseMove

             If e.Button = MouseButtons.Left Then
                startpoint = New Point(e.X, e.Y)
                Invalidate()
```

```
        ElseIf e.Button = MouseButtons.Right Then
            endpoint = New Point(e.X, e.Y)
            Invalidate()
        End If
    End Sub

    Private Sub Form1_Paint(ByVal sender As Object, _
           ByVal e As System.Windows.Forms.PaintEventArgs) _
           Handles MyBase.Paint

        Dim g As Graphics = e.Graphics
        g.DrawBezier(_
           Pens.Red, startpoint, control1, control2, endpoint)
    End Sub
```

3.10. Советы и рекомендации

Попробуйте расширить возможности редактора Безье из *разд. 3.9*, управляя внешним видом кривой Безье не только через начальную и конечную точки кривой, но и через две контрольные промежуточные точки.

Глава 4

Кривые

Рисование кривых линий — это очень увлекательное занятие. Фигуры, образуемые кривыми линиями, завораживают взор. Некоторые кривые получили свои имена. Изучением кривых занимались многие видные ученые. Кривые бывают простыми и сложными. Рассмотрим их поподробнее. Прежде всего надо отметить, что функции рисования в VB .NET реализованы через методы класса `Graphics`. Как правило, данные методы начинаются с префиксов `Draw` или `Fill`. В этих методах необходимо использовать объекты `Pen`, который отвечает за обрисовку линий, и `Brush`, используемый для заливки фигур.

Кроме того, при рисовании кривых используются тригонометрические методы класса `Math`. Программисты, перешедшие с VB 6, должны обратить внимание на данный класс. Теперь все тригонометрические функции, которые существовали в Visual Basic, содержатся именно там. Необходимо также помнить, что все вычисления идут в радианах, а не в градусах. Кроме того, класс `Math` уже содержит встроенную константу числа Пи. Теперь нет необходимости писать свою константу типа:

```
Const Pi = 3.14
```

Можно просто использовать в каком-нибудь выражении:

```
Math.PI
```

Теперь мы готовы перейти к практическим примерам.

4.1. Метод *DrawLines*

Для рисования кривых линий часто используется метод `DrawLines`. Этот метод способен нарисовать множество очень коротких отрезков, которые сливаются в кривую по заданной формуле. Один из перегруженных методов `DrawLines` класса `Graphics` выглядит так:

```
Sub DrawLines(ByVal pn As Pen, ByVal aptf as PointF( ))
```

Первый параметр задает используемое перо, а второй — содержит массив координат с плавающей точкой. Надо отметить, что данный метод может

содержать несколько сотен тысяч координат. Чем больше структур `PointF` вы передадите методу, тем более гладкой получится кривая. Метод обладает высокой производительностью, и вывод даже миллиона коротких линий на экран займет доли секунды.

4.1.1. Синусоида

Из школьного курса математики вам должна быть известна формула, необходимая для рисования синусоиды:

```
y=Sin(x)
```

Вот как может выглядеть пример (проект `Curves`), написанный на VB .NET (листинг 4.1).

Листинг 4.1. Рисование синусоиды

```
'--------------------------------------
' Синусоиды © 2004 Александр Климов
'--------------------------------------
    Private Sub butSinusoid_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles butSinusoid.Click
        Dim g As Graphics = CreateGraphics()

        Dim cx As Integer
        Dim cy As Integer
        cx = MyBase.Width
        cy = MyBase.ClientSize.Height / 2

        Dim ptf(cx - 1) As PointF
        Dim cW As Integer

        ' Число волн
        cW = txtWaves.Text

        ' Очистим форму
        g.Clear(Me.BackColor)

        Dim i As Integer
        For i = 0 To cx - 1
            ptf(i).X = i
```

```
            ptf(i).Y = CSng(_
                (cy / 2) * (1 - Math.Sin(i * cW * Math.PI / (cx - 1))))
        Next i
        g.DrawLines(Pens.Red, ptf)
        g.Dispose()
    End Sub
```

Разберем код поподробнее. В первой строчке мы создаем объект `Graphics`, который позволит нам рисовать кривые линии:

```
Dim g As Graphics = CreateGraphics()
```

Далее мы объявляем две переменные `cx` и `cy`, в которых будет содержаться диапазон допустимых значений. Эти значения я подобрал опытным путем, следя за тем, чтобы синусоида не выходила за пределы формы. В переменной `ptf` содержится массив координат точек. Для дополнительного удобства я добавил на форму текстовое поле, в котором можно будет задавать число волн, генерируемых синусоидой:

```
cW = txtWaves.Text    ' число волн
```

После чего мы запускаем цикл. Основная работа по выводу кривой на экран содержится в строчке

```
ptf(i).Y = CSng((cy / 2) * (1 - Math.Sin(i * cW * Math.PI / (cx - 1))))
```

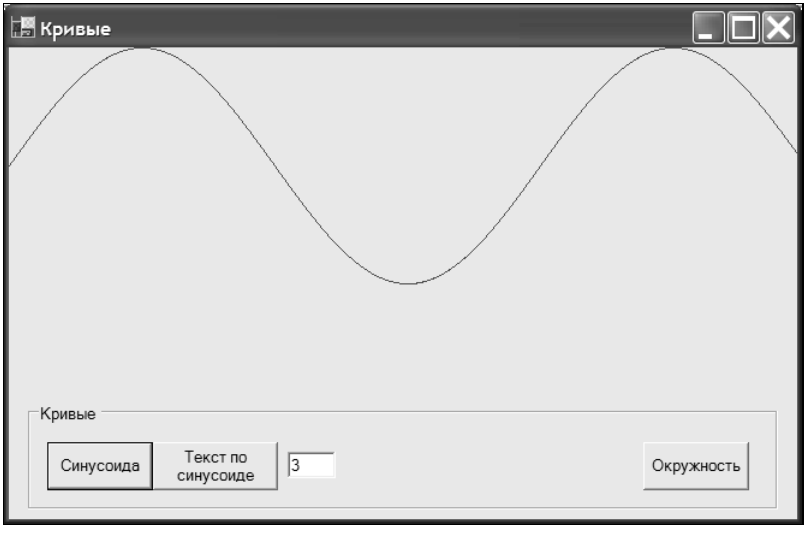

Рис. 4.1. Вывод синусоиды на экран

На первый взгляд эта строка не имеет ничего общего с формулой синусоиды. Но на самом деле все дополнительные переменные, используемые в данной строке, смещают или сплющивают синусоиду. Вы можете самостоятельно поиграть с различными значениями. В зависимости от ваших вводимых значений будут меняться высота волн синусоиды, их число и расположение на экране (рис. 4.1).

4.1.2. Текст вдоль синусоиды

То, что мы научились выводить синусоиду на экран — это хорошо, но вряд ли имеет практическую ценность. Тогда зачем мы ее рисовали? На самом деле синусоида очень широко используется в анимационных эффектах. Приглядитесь, например, к развевающемуся флагу. Если вы посмотрите внимательно на края флага, то увидите, что они имеют форму синусоиды. Кроме того, вдоль синусоиды можно рисовать не только линии, но и различные объекты и тексты. В самом деле, если метод `DrawString` использует в своих параметрах координаты символа, то почему бы не использовать эту возможность! Поместим на форму еще одну кнопку с именем `butSinText`. После небольшой модификации предыдущего примера мы получим следующий код (листинг 4.2).

Листинг 4.2. Вывод текста вдоль синусоиды

```
    Private Sub butSinText_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles butSinText.Click
        ' Процедура вывода текста вдоль синусоиды
        Dim g As Graphics = CreateGraphics()
        Dim cx As Integer
        Dim cy As Integer
        cx = MyBase.Width
        cy = MyBase.ClientSize.Height / 2

        Dim ptf(cx - 1) As PointF
        ' Число волн
        Dim cW As Integer
        cW = txtWaves.Text

        ' Очистим форму
        g.Clear(Me.BackColor)

        Dim i As Integer
        For i = 0 To cx - 1 Step 10
```

```
            Dim br As New SolidBrush(Color.RoyalBlue)
            Dim title As String = _
              "Занимательное программирование. Текст располагается вдоль
               синусоиды"
            Dim ch As Char

            If i >= (title.Length) * 10 Then
                ch = " "
            Else
                ch = title.Chars(i / 10)
            End If

            ptf(i).X = i
            ptf(i).Y = CSng(_
                (cy / 2) * (1 - Math.Sin(i * cW * Math.PI / (cx - 1))))
            g.DrawString(ch, Me.Font, br, ptf(i))
        Next i

        g.Dispose()
    End Sub
```

Прежде всего мы изменили шаг в цикле:

```
For i = 0 To cx - 1 Step 10
```

Иначе символы текста будут просто налезать друг на друга. Затем мы ввели переменную, которая будет содержать текст для вывода вдоль синусоиды:

```
Dim title As String = "Занимательное программирование. Текст
располагается вдоль синусоиды"
```

Теперь необходимо написать небольшое условие для нашего текста. Во-первых, мы объявим новую переменную `ch` для отдельных символов нашего текста. Во-вторых, мы проследим, чтобы счетчик не вышел за допустимые пределы. Если мы выходим за диапазон допустимых значений, который ограничен длиной текста, то просто выводим пустой символ (пробел).

```
Dim ch As Char

            If i >= (title.Length) * 10 Then
                ch = " "
            Else
                ch = title.Chars(i / 10)
 End If
```

Вместо вызова метода `DrawLines` мы теперь вызовем метод `DrawString`, где в первом параметре используем поочередно все символы заданного текста:

```
g.DrawString(ch, Me.Font, br, aptf(i))
```

Результат вы можете увидеть на рис. 4.2.

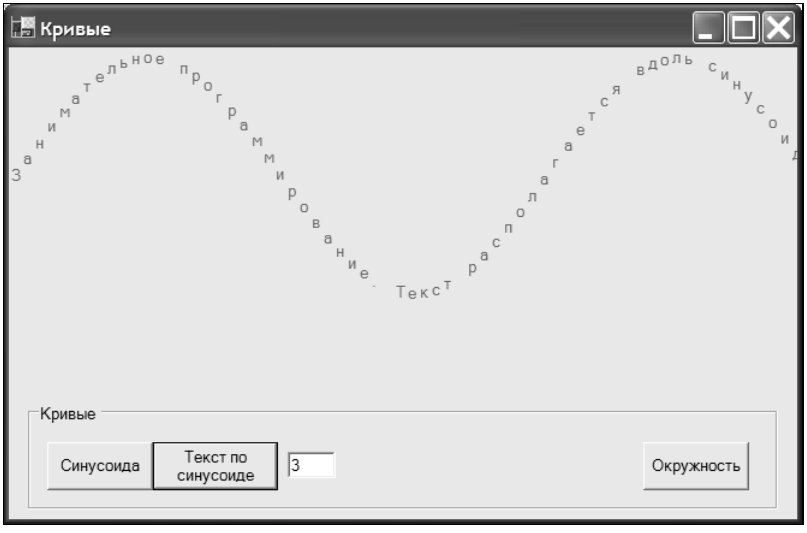

Рис. 4.2. Вывод текста вдоль синусоиды

4.2. Криволинейные системы координат

Приведенные только что примеры слишком просты, так как формула синусоиды примитивна. Но есть более сложные кривые, которые вывести на экран по обычным формулам гораздо сложнее. К примеру, формула для окружности выглядит так:

```
x2 + y2 = r2
```

Если вы попытаетесь вывести график данной функции, то столкнетесь с целым рядом проблем. Во-первых, вам придется извлекать квадратный корень. Во-вторых, надо будет следить за переменной `x`, чтобы не выйти за допустимый диапазон значений. В-третьих, сама функция очень нелинейная. При значениях `x`, близких к 0, точки окружности будут находиться близко друг к другу. При возрастании значений `x` изменения резко увеличиваются, что приведет к разрывам в рисовании окружности.

Здесь на помощь приходит использование других систем координат. Кроме привычной нам прямоугольной декартовой системы координат в математике

используются и другие способы задания положения точки на плоскости. Наиболее известны *полярные*, *цилиндрические* и *сферические* системы координат. Все эти системы родственны. В них присутствует центральная точка, называемая *полюсом*, от которого расходятся концентрические окружности (полярная система координат), цилиндры (цилиндрическая система) или сферы (сферические координаты). Положение точки определяется при помощи луча, выходящего из полюса и пересекающего в заданном месте соответствующую окружность, цилиндр или сферу.

Перечисленные криволинейные системы координат идеально приспособлены для отображения форм, построенных вокруг единой центральной точки. Фигуры, образуемые в криволинейных координатах, описываются достаточно простыми и лаконичными математическими выражениями и обладают неповторимой эстетической привлекательностью. Они ассоциируются с формами, которые мы видим в живой природе, — цветами, бабочками, ракушками и т. д.

4.2.1. Полярные координаты

В полярной системе координат положение точки определяется полярным радиусом `R` и углом `theta` (изменяемым от 0 до 2π радиан), образуемым полярным радиусом с полярной осью. Если в декартовой системе координат предельно простое выражение `y = kx` определяет прямую линию, то это же выражение, переписанное в форме `R = k * theta`, уже превращается в спираль. Фигуры в полярных координатах образуются как след конца бегающего по кругу полярного радиуса переменной длины. Длина полярного радиуса определяется величиной угла, который в данный момент времени он образует с полярной осью. Для того чтобы перейти от полярных координат к декартовой системе, используют параметрические уравнения. В этом случае формула для рисования окружности будет выглядеть как:

```
X(theta) = R * Cos (theta)
Y(theta) = R * Sin(theta)
```

где `theta` — угол, изменяющийся от 0 до 2π радиан, а `R` является радиусом окружности. Если переменной `R` присваивать разные значения, то мы получим сплюснутый или вытянутый эллипс. Таким образом, формулы кривых, записанные в полярной системе координат, вычисляются гораздо проще, чем в декартовой. Перейдем к практике. Вот код для кнопки, рисующей окружность (листинг 4.3).

Листинг 4.3. Рисование окружности

```
    Private Sub butCircle_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles butCircle.Click
```

```
    ' Рисование окружности своими силами
    Dim g As Graphics = CreateGraphics()
    Dim cx, cy As Integer
    cx = Me.ClientSize.Width / 3
    cy = Me.ClientSize.Width / 3
    Dim iNumPoints As Integer = 10000
    Dim ptf(iNumPoints) As PointF
    Dim i As Integer
    Dim redPen As New Pen(Color.Red)

    ' Очистим форму
    g.Clear(Me.BackColor)

    ' Переместим систему координат
    g.TranslateTransform(200, 150)

    For i = 0 To iNumPoints
        Dim theta As Double = i * 2 * Math.PI / iNumPoints
        ptf(i).X = CSng((cx) / 2 * (Math.Cos(theta)))
        ptf(i).Y = CSng((cy) / 2 * (Math.Sin(theta)))
    Next i
    g.DrawLines(redPen, ptf)
    g.Dispose()
End Sub
```

В данном примере я присвоил переменной `iNumPoints` значение 10 000. Думаю, что десяти тысяч точек вполне достаточно для вывода окружности на экран. Кое-кто из читателей, возможно, спросит, а зачем рисовать окружность таким способом, если уже есть встроенный метод `DrawEllipse`? На самом деле между двумя этими способами есть небольшая разница. Метод `DrawEllipse` рисует готовую окружность, мы же рисуем множество очень маленьких линий по заданным координатам, которые складываются в окружность. В *разд. 4.1.2* я уже показал, как можно выводить текст вдоль некоторой линии. Такой же подход можно использовать и для вывода по окружности. Но главное, существуют кривые, для рисования которых не существует готовых методов в Visual Basic .NET 2003. О таких кривых и пойдет далее речь.

4.2.2. Четырехлистный клевер

Полярные координаты позволяют рисовать намного более сложные и красивые фигуры. Например, можно нарисовать четырехлистный клевер. Его формула выглядит как `R = Cos (2*theta)`, где угол `theta` меняется от 0 до 2π радиан (от 0 до 360 градусов). Немного переделаем пример, где мы рисовали окружность, и получим проект `Curves2` (листинг 4.4).

Листинг 4.4. Рисование четырехлистного клевера

```
'--------------------------------------
' Четырехлистный клевер © 2004 Александр Климов
'--------------------------------------
    Private Sub butClever_Click(ByVal sender As System.Object, _
                ByVal e As System.EventArgs) Handles butClever.Click
        Dim g As Graphics = CreateGraphics()
        Dim cx, cy As Integer
        cx = Me.ClientSize.Width / 3
        cy = Me.ClientSize.Width / 3
        Dim iNumPoints As Integer = 2 * (cx + cy)
        Dim aptf(iNumPoints) As PointF

        g.Clear(BackColor)

        Dim i As Integer
        For i = 0 To iNumPoints
            Dim theta As Double = i * 2 * Math.PI / iNumPoints
            Dim r As Double = Math.Cos(2 * theta)
            aptf(i).X = CSng(_
              (cx - 1) / 2 * (1 + Math.Cos(theta) * Math.Cos _
              txtTheta.Text * theta)))
            aptf(i).Y = CSng(_
              (cy - 1) / 2 * (1 + Math.Sin(_
               theta) * Math.Cos(txtTheta.Text * theta)))
        Next i
        g.DrawLines(Pens.Red, aptf)
        g.Dispose()
    End Sub
```

В этом примере мы использовали текстовое поле с именем `txtTheta` с предустановленным свойством `Text = 2`. Попробуйте поиграть с этим значением. Вы увидите, как программа рисует красивые цветы (рис. 4.3).

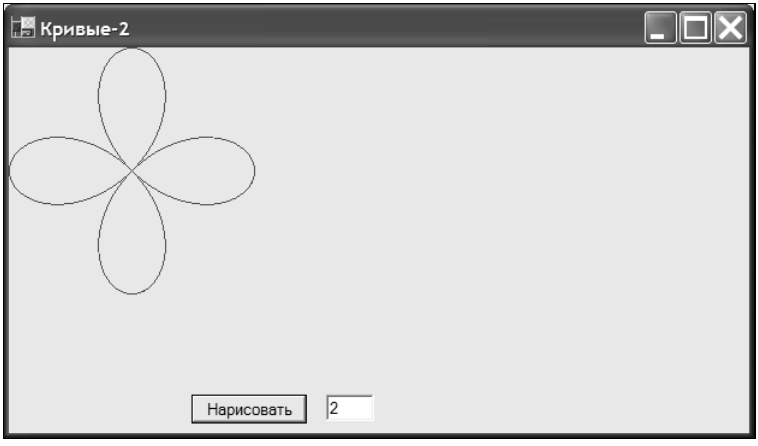

Рис. 4.3. Рисование четырехлистного клевера

4.3. Семейство кривых линий

На протяжении многих лет ученые собирали информацию о формулах, рисующих разные фигуры. Многие фигуры получили свои названия. Список таких названий внушителен. Вот только часть из них:

- дельтоида;
- астроида;
- кардиоида;
- лимакона (улитка Паскаля);
- спираль Архимеда;
- логарифмическая спираль;
- кохлеоида;
- строфоида;
- нефроида.

Прежде чем приступить к рисованию различных кривых, надо создать заготовку для проекта. Тогда нам будет легче рассматривать получаемые кривые. Проект для рисования кривых носит название `DrawCurves`. Запустим новый

проект и сразу добавим в него новый модуль `Transformation.vb`, в котором будет осуществляться перенос отсчета координат в центр формы (листинг 4.5). Так будет гораздо удобнее, чем отталкиваться от верхней левой точки формы.

Листинг 4.5. Модуль Transformation.vb

```
 '--------------------------------------
' Рисование кривых © 2004 Александр Климов
'--------------------------------------
Imports System.Drawing.Drawing2D
Module Transformation
    ' Процедура преобразования
    ' глобальных координат wxmin, wymin, wxmax, wymax
    ' в координаты устройства dxmin, dymin, dxmax, dymax
    Public Sub MapGraphicsWindow(ByVal g As Graphics, _
               ByVal wxmin As Single, ByVal wymin As Single, _
               ByVal wxmax As Single, ByVal wymax As Single, _
               ByVal dxmin As Single, ByVal dymin As Single, _
               ByVal dxmax As Single, ByVal dymax As Single)

        ' Возвращаемся в исходное состояние
        g.ResetTransform()

        ' Преобразуем глобальные координаты центра
        g.TranslateTransform(-(wxmax + wxmin) / 2, _
                             -(wymax + wymin) / 2, _
                             MatrixOrder.Append)

        ' Масшабируем
        Dim sx As Single
        sx = CSng((dxmax - dxmin) / (wxmax - wxmin))
        Dim sy As Single
        sy = CSng((dymax - dymin) / (wymax - wymin))
        g.ScaleTransform(sx, sy, MatrixOrder.Append)

        ' Преобразуем в центр формы
        g.TranslateTransform((dxmax + dxmin) / 2, _
```

```
                                    (dymax + dymin) / 2, _
                                    MatrixOrder.Append)
    End Sub
End Module
```

4.3.1. Полигон для опытов

После создания модуля возвращаемся в главную форму. Так как кривых линий очень много, то я расскажу о наиболее известных из них. Рисование будет происходить через пункты меню. Также предусмотрим возможность рисования координатной сетки. Основная заготовка для формы выглядит так (листинг 4.6).

Листинг 4.6. Заготовка для вывода кривых линий

```
'--------------------------------------
' Рисование кривых © 2004 Александр Климов
'--------------------------------------
Imports System.Math

    Private Sub DrawAxes(ByVal g As Graphics)
        ' Рисуем координатные оси
        Dim x As Double
        Dim y As Double

        ' Черное перо для координатных осей
        Dim p_axes As New Pen(Color.Black, 0)

        g.DrawLine(p_axes, -2, 0, 2, 0)
        ' Шкала по оси X
        For x = -2 To 2 Step 0.5
            g.DrawLine(p_axes, CSng(x), CSng(-0.1), CSng(x), CSng(0.1))
        Next x
        ' Шкала по оси Y
        g.DrawLine(p_axes, 0, -2, 0, 2)
        For y = -2 To 2 Step 0.5
            g.DrawLine(p_axes, CSng(-0.1), CSng(y), CSng(0.1), CSng(y))
        Next y
    End Sub
```

```
    Private Sub mnuAxes_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles mnuAxes.Click
        ' Состояние галочки
        mnuAxes.Checked = Not mnuAxes.Checked
    End Sub
```

4.3.2. Четырехлистный клевер — второй способ

Шаблон готов. Можно приступать к рисованию кривых. Для этого достаточно добавить новый пункт меню и написать процедуру для самой кривой. Начнем с вывода знакомого нам четырехлистного клевера. Его формула выглядит так:

```
R = Cos (2*theta)
```

Пишем отдельную процедуру для построения четырехлистника и вызываем ее через событие `Click` пункта меню `mnuFourLeaf` (листинг 4.7).

Листинг 4.7. Рисование четырехлистника

```
    Private Sub DrawFourLeaf(ByVal g As Graphics)
        ' Рисуем четырехлистник
        Dim x As Double
        Dim y As Double
        Dim old_x As Double
        Dim old_y As Double
        Dim r As Double
        Dim t As Double
        Dim dt As Double
        Dim iFromTxt As Integer = CInt(txtSettings.Text)

        ' Рисуем кривую
        t = 0
        dt = PI / 100
        old_x = 0
        old_y = 0
        Dim p_curve As New Pen(Color.Blue, 0)
        Do While t <= 2 * PI
            ' Формула четырехлистника r = CSng(1 * Cos(2 * t))
            r = CSng(1 * Cos(iFromTxt * t))
            x = r * Cos(t)
```

```
            y = r * Sin(t)
            g.DrawLine(p_curve, CSng(old_x), CSng(old_y), _
                       CSng(x), CSng(y))
            old_x = x
            old_y = y
            t = t + dt
        Loop

    End Sub

    Private Sub mnuFourLeaf_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles mnuFourLeaf.Click

        Dim g As Graphics = CreateGraphics()

        MapGraphicsWindow(g, _
                  -2, -2, 2, 2, _
                  0, ClientSize.Height - 1, _
                  ClientSize.Width - 1, 0)

        ' Рисуем четырехлистник
        g.Clear(BackColor)
        If mnuAxes.Checked = True Then
            DrawAxes(g)
        End If

        DrawFourLeaf(g)

        g.Dispose()
    End Sub
```

Строго говоря, с помощью этого кода можно нарисовать не только четыре лепестка, но и три, и пять, и двадцать пять. Я предусмотрительно не стал использовать константу в формуле кривой, а заменил ее значением, которое берется из текстового поля

```
Dim iFromTxt As Integer = CInt(txtSettings.Text)
```

По умолчанию в редакторе свойств свойству `Text` текстового поля присвоено значение 2, которое и позволяет нарисовать четырехлистник. Попробуйте

во время выполнения программы поиграть с этими значениями. Вы получите целый букет цветов (рис. 4.4).

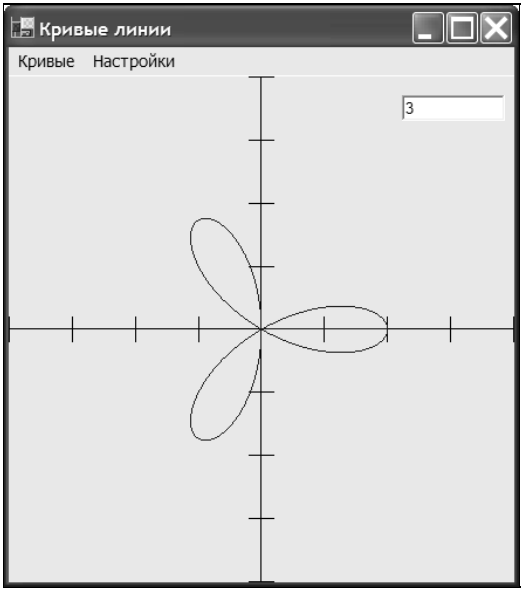

Рис. 4.4. Цветок при значении `txtSettings.Text=3`

4.3.3. Пируэты окружности

Возьмем теперь одну окружность и поместим ее внутрь другой. Все кривые, которые будет вычерчивать точка на окружности, катящейся внутри другой окружности, будут относиться к семейству *гипоциклоид* (от греч. *гипо* — под, внизу и *киклоидес* — кругообразный). Как вы думаете, какую траекторию опишет точка окружности, которая катится внутри другой окружности? Как это ни странно звучит, но она может быть даже прямой! Для этого радиус внутренней окружности должен быть в два раза меньше радиуса внешней. Первым это заметил и описал Николай Коперник. Если же радиус внутренней окружности меньше радиуса большой окружности в три раза, то точка опишет *кривую Штейнера* (дельтоиду).

Дельтоида

Формула для дельтоиды (рис. 4.5) выглядит так:

```
x = 2 * a * cos(theta) + a * cos(2 * theta)
y = 2 * a * sin(theta) - a sin(2 * theta)
```

Переменная a является радиусом маленькой окружности, которая в три раза меньше большой окружности, внутри которой она вращается (листинг 4.8).

Листинг 4.8. Рисование дельтоиды

```
    Private Sub DrawDeltoid(ByVal g As Graphics)
        ' Рисуем дельтоиду
        Dim x As Double
        Dim y As Double
        Dim old_x As Double
        Dim old_y As Double
        Dim r As Double
        Dim t As Double
        Dim dt As Double
        Dim iFromTxt As Integer = CInt(txtSettings.Text)

        t = 0
        dt = PI / 100
        old_x = 0
        old_y = 0
        Dim p_curve As New Pen(Color.Blue, 0)
        Do While t <= 2 * PI
            ' Формула для дельтоиды
            'x = 2 a cos(theta) + a cos(2 theta)
            'y = 2 a sin(theta) - a sin(2 theta)
            x = 1 * Cos(t) + 0.5 * Cos(2 * t)
            y = 1 * Sin(t) - 0.5 * Sin(2 * t)
            g.DrawLine(p_curve, CSng(old_x), CSng(old_y) _
                       CSng(x), CSng(y))
            old_x = x
            old_y = y
            t = t + dt
        Loop
    End Sub

    Private Sub mnuDeltoid_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles mnuDeltoid.Click
        Dim g As Graphics = CreateGraphics()
```

```
    MapGraphicsWindow(g, _
                      -2, -2, 2, 2, _
                      0, ClientSize.Height - 1, _
                      ClientSize.Width - 1, 0)
    g.Clear(BackColor)
    If mnuAxes.Checked = True Then
        DrawAxes(g)
    End If

    DrawDeltoid(g)
    g.Dispose()
End Sub
```

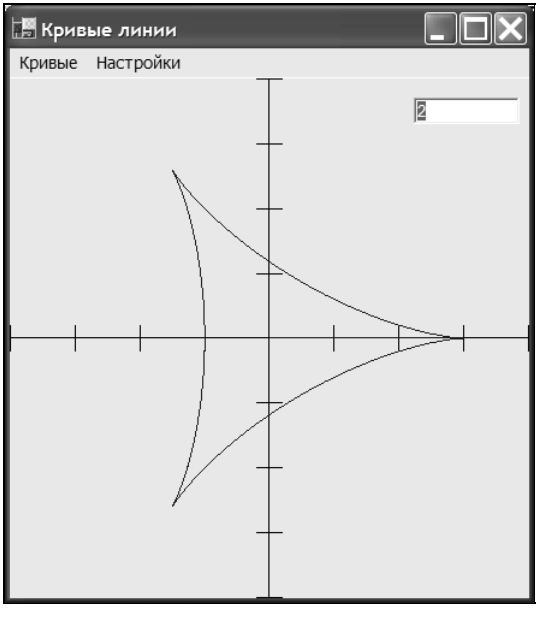

Рис. 4.5. Рисование дельтоиды

Астроида

Уменьшив радиус внутренней окружности теперь в четыре раза по сравнению с большой окружностью, мы получим *астроиду* (рис. 4.6). Название кривой произошло от греческого слова *астрон* — звезда. Данная кривая внешне напоминает стилизованное изображение звезды. Теперь вы знаете,

почему наука о звездах называется астрономией. Формула для астроиды выглядит так:

```
x = a* cos(t)^3
y = a* sin(t)^3
```

где коэффициент `a` влияет на вытянутость фигуры. Добавляем в проект новый пункт `mnuAstroid` и пишем код (листинг 4.9).

Листинг 4.9. Рисование астроиды

```
Private Sub DrawAstroid(ByVal g As Graphics)
    ' Рисуем астроиду
    Dim x As Double
    Dim y As Double
    Dim old_x As Double
    Dim old_y As Double
    Dim r As Double
    Dim t As Double
    Dim dt As Double
    Dim iFromTxt As Integer = CInt(txtSettings.Text)

    t = 0
    dt = PI / 100
    old_x = 0
    old_y = 0
    Dim p_curve As New Pen(Color.Blue, 0)
    Do While t <= 2 * PI
        ' Формула для астроиды
        'x = a cos(theta)^3
        'y = a sin(theta)^3
        x = iFromTxt * Cos(t) ^ 3
        y = iFromTxt * Sin(t) ^ 3
        g.DrawLine(p_curve, CSng(old_x), CSng(old_y), _
                   CSng(x), CSng(y))
        old_x = x
        old_y = y
        t = t + dt
    Loop

End Sub
```

```
    Private Sub mnuAstroid_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles mnuAstroid.Click
        Dim g As Graphics = CreateGraphics()

        MapGraphicsWindow(g, _
                  -2, -2, 2, 2, _
                  0, ClientSize.Height - 1, _
                  ClientSize.Width - 1, 0)
        g.Clear(BackColor)
        If mnuAxes.Checked = True Then
            DrawAxes(g)
        End If

        DrawAstroid(g)
        g.Dispose()
    End Sub
```

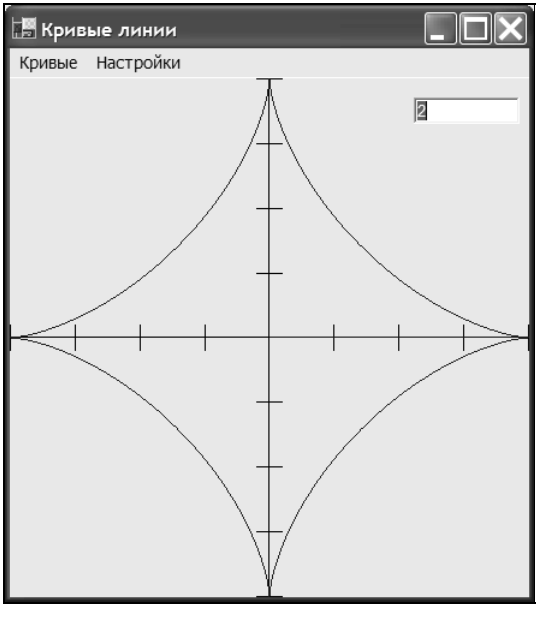

Рис. 4.6. Рисование астроиды

4.3.4. Эпициклоиды

Рассмотрим другой случай. Будем вращать окружность не внутри другой (опорной) окружности, а по ее внешней стороне. Теперь все получаемые

кривые будут относиться к семейству *эпициклоид* (от греческого *эпи* — на, над). К таким фигурам относятся *кардиоида* и *улитка Паскаля*.

Кардиоида

Если использовать две окружности с одинаковыми радиусами и вращать одну вокруг другой, то взятая на окружности точка выпишет фигуру, отдаленно напоминающую сердце (рис. 4.7). Такая кривая получила название кардиоида (от греческого слова *кардиа* — сердце). По мнению математиков получаемая кривая действительно напоминает сердце. Формула, рисующая кардиоиду, выглядит следующим образом:

```
r = 2a(1 + cos(theta))
```

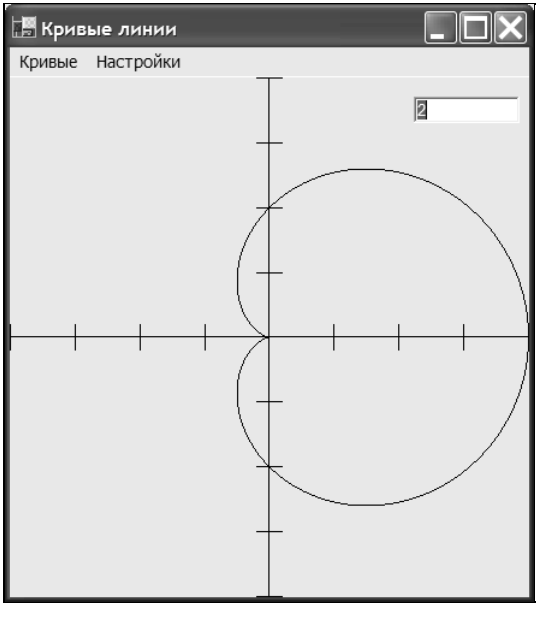

Рис. 4.7. Рисование кардиоиды

Переменная `a` является радиусом для этих двух окружностей. В нашем примере радиус окружности принят за 0,5 условных единиц. Поэтому формула в редакторе кода примет следующий код:

```
r = CSng(1 * (1 + Cos(t)))
```

Нам осталось только добавить новый пункт в меню и написать код для процедур `DrawCardioid` и `mnuCardioid.Click` (листинг 4.10).

Листинг 4.10. Рисование кардиоиды

```
    Private Sub DrawCardioid(ByVal g As Graphics)
        ' Рисуем кардиоиду
        Dim x As Double
        Dim y As Double
        Dim old_x As Double
        Dim old_y As Double
        Dim r As Double
        Dim t As Double
        Dim dt As Double

        t = 0
        dt = PI / 100
        old_x = 0
        old_y = 0
        Dim p_curve As New Pen(Color.Black, 0)
        Do While t <= 2 * PI
            'кардиодида r = 2a(1 + cos(theta)),
            'где a - радиус двух окружностей
            r = CSng(1 * (1 + Cos(t)))
            x = r * Cos(t)
            y = r * Sin(t)
            g.DrawLine(p_curve, CSng(old_x), CSng(old_y), _
                       CSng(x), CSng(y))
            old_x = x
            old_y = y
            t = t + dt
        Loop

    End Sub

    Private Sub mnuCardioid_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles mnuCardioid.Click
        Dim g As Graphics = CreateGraphics()

        MapGraphicsWindow(g, _
                  -2, -2, 2, 2, _
                  0, ClientSize.Height - 1, _
                  ClientSize.Width - 1, 0)
```

```
        g.Clear(BackColor)
        If mnuAxes.Checked = True Then
            DrawAxes(g)
        End If

        DrawCardioid(g)

        g.Dispose()
    End Sub
```

Лимакона, или улитка Паскаля (Limacon of Pascal)

А что произойдет, если брать точку не самой катящейся окружности, а внутри ее, сместив в сторону от центра? Тогда мы получим кривую, получившуюся название *улитка Паскаля,* или *лимакона* (рис. 4.8). Эта кривая была открыта французским математиком Этьеном Паскалем (отцом знаменитого ученого Блеза Паскаля). Улитку Паскаля можно нарисовать по формуле:

```
r = b + 2a cos(theta)
```

Рассмотренная выше кривая кардиоида является частным случаем лимаконы. В этом легко убедиться, если установить равенство `b = 2a`. Так как рисование кривой фигуры не слишком отличается от предыдущих, то я приведу только фрагмент кода (листинг 4.11). Полный листинг вы сможете просмотреть в проекте, представленном на компакт-диске.

Листинг 4.11. Рисование улитки Паскаля

```
        Do While t <= 2 * PI
            ' Формула для лимаконы
            'r = b + 2a cos(theta)
            r = CSng(0.3 + 1 * (Cos(t)))
            x = r * Cos(t)
            y = r * Sin(t)
            gr.DrawLine(p_curve, CSng(old_x), CSng(old_y) _
                        CSng(x), CSng(y))
            old_x = x
            old_y = y
            t = t + dt
        Loop
```

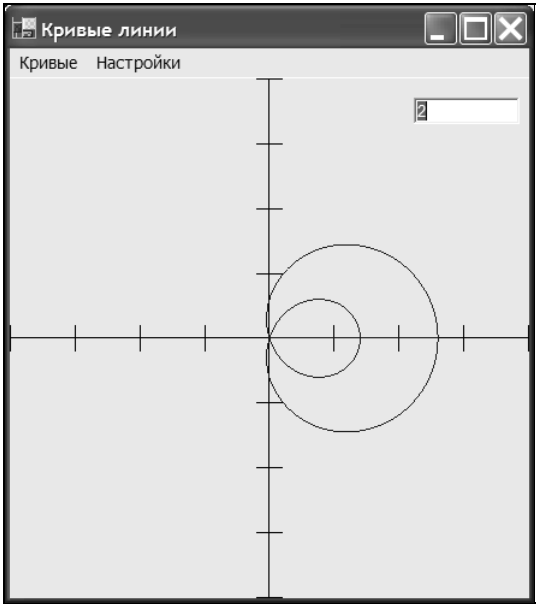

Рис. 4.8. Рисование улитки Паскаля

Другие кривые

Вкратце приведу названия и формулы для других кривых. Рекомендую посмотреть все эти кривые в проекте `DrawCurves` на прилагаемом компакт-диске.

- Нефроида — кривая, описываемая окружностью при вращении вокруг другой окружности в два раза больше по диаметру. Формула нефроиды:

```
x = a (3 cos(theta) - cos(3 theta))
y = a (3 sin(theta) - sin(3 theta))
```

- Лемниската — кривая, напоминающая знак бесконечности. Формула лемнискаты:

```
r^2 = a^2 cos(2 theta)
```

4.3.5. Спираль Архимеда

В повседневной жизни мы не раз видели спиралевидные фигуры. Если вы любите мыться, то, очевидно, замечали, как вода образует воронку, вытекая из ванны. Спиралями интересовался великий греческий мыслитель Архимед. Как вы, вероятно, знаете, свое знаменитое восклицание "Эврика!" гре-

ческий ученый произнес именно в ванне (правда к спиралям это никакого отношения не имело).

Если представить муравья, передвигающегося по секундной стрелке часов с постоянной скоростью, то его траектория и будет спиралью Архимеда. Она была названа в его честь потому, что он использовал ее свойства в задаче о трисекции угла. Формула спирали Архимеда в полярных координатах выглядит следующим образом:

```
r = k*theta
```

Используя пример с клевером (см. проект `Curves2`, листинг 4.4), модифицируем немного код применительно к спирали — проект `Spiral` (листинг 4.12):

```
aptf(i).X = CSng((cx) / 10 * (8 + Math.Cos(theta) * 0.09 * theta))
aptf(i).Y = CSng((cy) / 10 * (8 + Math.Sin(theta) * 0.09 * theta))
```

Листинг 4.12. Спираль Архимеда

```
'--------------------------------------
' Спирали © 2004 Александр Климов
'--------------------------------------
Private Sub butArchim_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles butArchim.Click
        Dim g As Graphics = CreateGraphics()
        Dim cx, cy As Integer
        cx = ClientSize.Width / 2
        cy = ClientSize.Height / 2
        Const iNumRevs As Integer = 20
        Dim iNumPoints As Integer = 5000
        Dim aptf(iNumPoints) As PointF
        Dim theta, rScale As Double
        Dim i As Integer
        g.Clear(BackColor)

        For i = 0 To iNumPoints
            theta = i * 2 * Math.PI / (iNumPoints / iNumRevs)
            rScale = i / iNumPoints
            aptf(i).X = CSng(cx * (1 + rScale * Math.Cos(theta)))
            aptf(i).Y = CSng(cy * (1 + rScale * Math.Sin(theta)))
        Next i
```

```
        Dim blackPen As New Pen(Color.Black, 4)
        g.DrawLines(blackPen, aptf)
    End Sub
```

Результат вы можете видеть на рис. 4.9. Как вы помните, метод `DrawLines` рисует множество коротких отрезков, образующих кривую линию. А что если число таких отрезков резко уменьшить? Вы получите очень интересные результаты. Попробуйте присвоить переменной `iNumPoints` число поменьше, ну, скажем, 15. Вы получите своеобразную треугольную спираль. Если вы посчитаете число прямых отрезков этого треугольника, то увидите, что оно совпадает со значением `iNumPoints`.

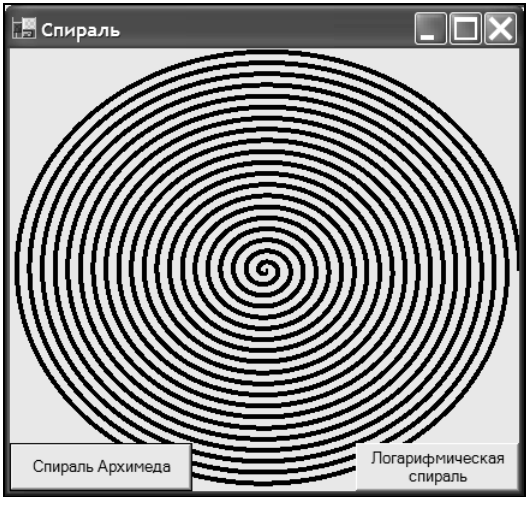

Рис. 4.9. Спираль Архимеда

4.3.6. Логарифмическая спираль

Рассмотрим другой случай. Пусть три муравья, находящиеся на равноудаленном расстоянии (вершины правильного треугольника), решили познакомиться друг с другом. Первый пошел ко второму, второй — к третьему, а третий — к первому. Путешествуя с одинаковой скоростью, муравьи всегда будут находиться в вершинах правильного треугольника, подобного исходному (только поменьше), описывая при этом дугу логарифмической спирали. Логарифмическая спираль выражается формулой:

```
R=a^theta
```

Подобие логарифмической спирали можно увидеть в витках раковины улитки. Свойства логарифмической спирали изучались многими учеными и нашли свое применение в различных отраслях промышленности. А швейцарский математик Якоб Бернулли завещал даже высечь ее на своей могиле. Используя приведенную формулу после перевода ее в полярную систему координат, мы получим следующий код (листинг 4.13).

Листинг 4.13. Логарифмическая спираль

```
Private Sub butLogSpiral_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles butLogSpiral.Click
        Dim g As Graphics = CreateGraphics()
        Dim cx, cy As Integer
        cx = ClientSize.Width / 50
        cy = ClientSize.Height / 50
        Const iNumRevs As Integer = 8
        Dim iNumPoints As Integer = 5000
        Dim aptf(iNumPoints) As PointF
        Dim theta, rScale As Double
        Dim i As Integer
        g.Clear(BackColor)

        For i = 0 To iNumPoints
            theta = i * 2 * Math.PI / (iNumPoints / iNumRevs)
            rScale = i / iNumPoints
            aptf(i).X = CSng(cx * (20 + Math.Cos(theta) * 1.07 ^ theta))
            aptf(i).Y = CSng(cy * (20 + Math.Sin(theta) * 1.07 ^ theta))
        Next i

        Dim blackPen As New Pen(Color.Black, 4)
        g.DrawLines(blackPen, aptf)
End Sub
```

4.3.7. Циклоиды

Мы уже увидели, какие получаются интересные фигуры, если точка лежит не на самой обегающей окружности, а внутри или снаружи ее. Все они относятся к общей группе *циклоид*. Попробуем воспроизвести несколько подобных кривых. Для этого нам понадобятся две параметрические функции и процедура, рисующая кривую. Один параметр из этой процедуры я решил

не фиксировать жестко программным путем, а брать из текстового поля. Попробуйте поиграть с этим значением, чтобы посмотреть на разные кривые — проект `Hypocycloid` (листинг 4.14).

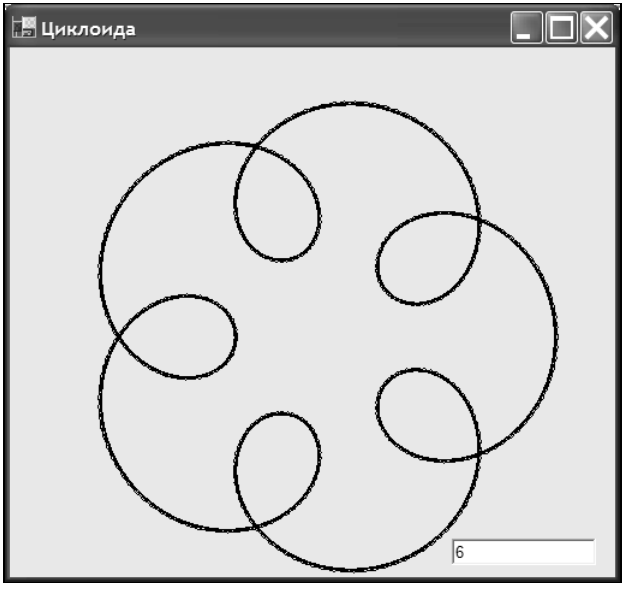

Рис. 4.10. Один из видов циклоиды

Листинг 4.14. Рисование циклоид

```
'--------------------------------------
' Циклоиды © 2004 Александр Климов
'--------------------------------------
   ' Процедура для рисования циклоидов
    Private Sub DrawCycloid(ByVal g As Graphics, _
           ByVal cx As Integer, ByVal cy As Integer _
           ByVal start_t As Single, ByVal stop_t As Single, _
           ByVal dt As Single)

        Dim num_pts As Integer
        Dim pts() As PointF
        Dim pt As Integer
        Dim t As Single
```

```
        Dim px As Single
        Dim py As Single

        num_pts = (stop_t - start_t) / dt
        ReDim pts(num_pts)

        ' Определяем центр для рисования.
        px = cx + X(start_t)
        py = cy + Y(start_t)

        ' Определяем все точки для кривой
        t = start_t
        For pt = 0 To num_pts - 1
            pts(pt).X = cx + X(t)
            pts(pt).Y = cy + Y(t)
            t += dt
        Next pt

        pts(num_pts) = pts(0)
        ' Соединяем все точки.
        g.DrawLines(New Pen(Color.Black), pts)
    End Sub

    ' Параметрическая функция X(t).
    Private Function X(ByVal t As Single) As Single
        X = (20000 * (2 * Cos(t) + Cos(t * txtParam.Text)) / 30) / 10
    End Function

    ' Параметрическая функция Y(t).
    Private Function Y(ByVal t As Single) As Single
        Y = (20000 * (2 * Sin(t) + Sin(t * txtParam.Text)) / 30) / 10
    End Function

    Private Sub Form1_Paint(ByVal sender As Object, ByVal e As
System.Windows.Forms.PaintEventArgs) Handles MyBase.Paint
        'Выводим при загрузке формы
        DrawCycloid(e.Graphics, _
            Me.Width \ 2, Me.Height \ 2, _
```

```
            0, 14 * PI, 0.1)
    End Sub

    Private Sub txtParam_TextChanged(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles txtParam.TextChanged
        Dim g As Graphics = CreateGraphics()
        g.Clear(Me.BackColor)
        DrawCycloid(g, _
                    Me.Width \ 2, Me.Height \ 2, _
                    0, 14 * PI, 0.1)
        g.Dispose()

    End Sub
```

4.3.8. Паутинка

Продолжим наши опыты с кривыми. Возьмем некоторую окружность, на которой с определенным шагом берутся точки, и каждую из этих точек соединим с точкой, сдвинутой по фазе какое-то число раз `N`. В этом случае точки пересечения хорд сольются в муаровый узор самых замысловатых форм. Например, при `N = 1` на экране не нарисуется ничего, так как начальные и конечные точки линий совпадают, зато при увеличении значения переменной `N` будут появляться фигуры с узлами, причем количество узлов равно `N - 1`. Самое интересное начинается при `N = 2`. В этом случае нарисуется уже изученная нами кардиоида. При `N = 3` получится нефроида с двумя узлами. Если `n - 1` является делителем числа 360, то картинка проявляет некоторую упорядоченность — проект `Pautina` (листинг 4.15).

Листинг 4.15. Создание узоров

```
'---------------------------------------
' Паутинка © 2004 Александр Климов
'---------------------------------------
    Private Sub butGetPautina_Click(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles butGetPautina.Click
        Static N As Double
        ' Координаты паутинки
        Dim xx As Integer = 350
        Dim yy As Integer = 270
```

```
        ' Радиус паутинки
        Dim R As Integer = 270
        N = TextBox1.Text
        Dim g As Graphics = CreateGraphics()
        g.Clear(BackColor)

        Dim i As Integer
        For i = 0 To 360 Step 2
            Dim T As Double
            T = i * PI / 180
            ' Координаты первых точек пары на окружности
            Dim x As Double
            x = R * System.Math.Cos(T)
            Dim y As Double
            y = R * System.Math.Sin(T)
            ' Координаты смещенных вторых точек пары
            Dim x2 As Double
            x2 = R * System.Math.Cos(N * T)
            Dim y2 As Double
            y2 = R * System.Math.Sin(N * T)

            'Соединяем точки
            g.DrawLine(Pens.Blue, _
                       CInt(x + xx), CInt(y + yy), _
                       CInt(x2 + xx), CInt(y2 + yy))
        Next i
    End Sub
```

В текстовом поле вы можете установить другие значения, чтобы посмотреть на получаемые узоры. А можно пойти другим путем и облегчить работу, поручив самой программе изменять параметр `N` через событие таймера. В этом случае у нас будет возможность наблюдать череду сменяющихся узоров. Зрелище интересное, напоминающее калейдоскоп (листинг 4.16).

Листинг 4.16. Создание узоров через таймер

```
    Private Sub butTimer_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles butTimer.Click
        Timer1.Start()
    End Sub
```

```
    Private Sub Timer1_Tick(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Timer1.Tick
        Static N As Double

        ' Координаты паутинки
        Dim xx As Integer = 350
        Dim yy As Integer = 270

        ' Радиус паутинки
        Dim R As Integer = 270

        N = N + 0.03
        Dim g As Graphics = CreateGraphics()
        g.Clear(BackColor)

        Dim i As Integer
        For i = 0 To 360 Step 2
            Dim T As Double
            T = i * PI / 180
            Dim x As Double
            x = R * System.Math.Cos(T)
            Dim y As Double
            y = R * System.Math.Sin(T)
            Dim x2 As Double
            x2 = R * System.Math.Cos(N * T)
            Dim y2 As Double
            y2 = R * System.Math.Sin(N * T)
            g.DrawLine(Pens.Blue, CInt(x + xx), CInt(y + yy), _
                       CInt(x2 + xx), CInt(y2 + yy))
        Next i
    End Sub
```

Если запустить таймер и смотреть продолжительное время на получаемые узоры, то можно увидеть очень красивые фигуры (рис. 4.11). Подобную программу так и хочется использовать в качестве хранителя экрана.

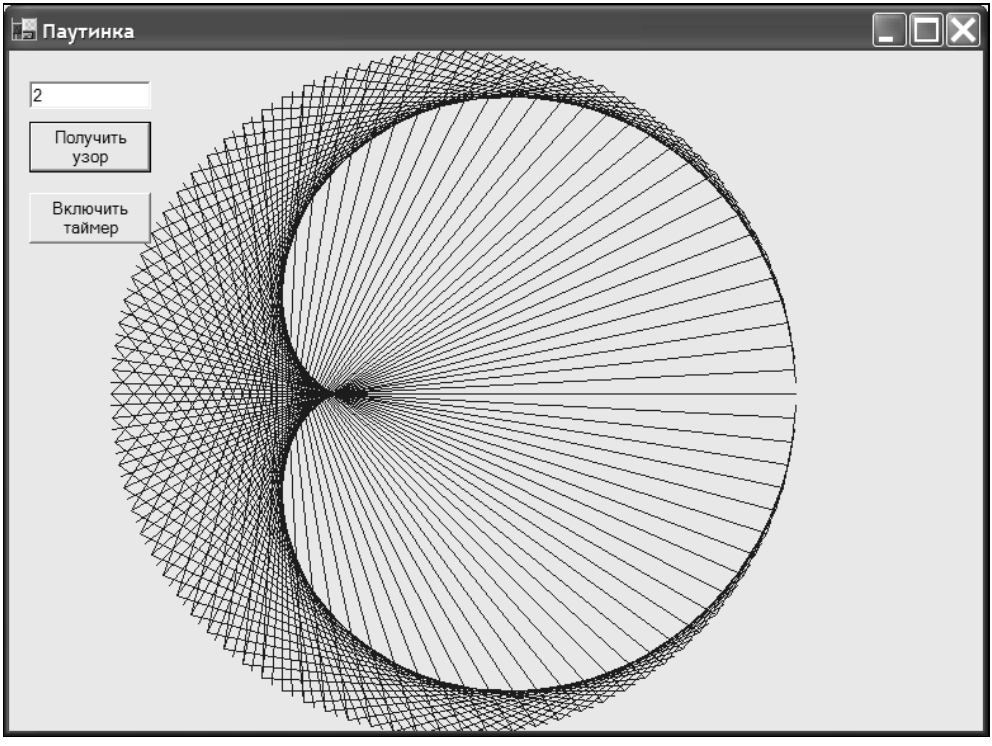

Рис. 4.11. Паутинка

4.4. Советы и рекомендации

Существует большое количество красивых кривых. Советую самостоятельно найти информацию о них, узнать их формулы и получить график кривой на вашем компьютере.

Глава 5

Градиентные заливки

Градиент — это плавный переход от одного цвета к другому. Поверхности с градиентной заливкой смотрятся очень эффектно. Раньше на форумах для VB-программистов тема создания градиентов была весьма популярной. Существовали даже коммерческие ActiveX-элементы управления, дающие возможность создания градиентных поверхностей. Суть создания градиента состояла в прорисовке тонких линий с постепенно изменяющимся в цикле цветом этих линий. Но в VB .NET возможности использования градиентов настолько мощны и разнообразны и одновременно настолько легки, что использование старых приемов из проектов VB 6.0 потеряло всякий смысл. В этой главе мы разберем несколько таких приемов.

5.1. Класс *LinearGradientBrush*

Класс `LinearGradientBrush` является производным от абстрактного класса `Brush` и служит для создания кисти, которая заполняет фигуры градиентным цветом вдоль прямой линии. Существует несколько перегруженных версий данного класса. Вот небольшой список из них:

- `Public Sub New(Point, Point, Color, Color)` — указываются начальная и конечная точки линейного градиента, а также начальный и конечный цвета градиента;
- `Public Sub New(Rectangle, Color, Color, LinearGradientMode)` — указываются ограничивающий прямоугольник для градиента, начальный и конечный цвет градиента и направление градиента;
- `Public Sub New(Rectangle, Color, Color, Single)` — указываются ограничивающий прямоугольник для градиента, начальный и конечный цвет градиента и угол направления градиента.

Для самого первого знакомства с этим классом создадим простой пример (проект `LinearGradientBrushSample`), рисующий круг с горизонтальной градиентной заливкой (листинг 5.1).

Листинг 5.1. Создание градиентной заливки

```
'--------------------------------------
' Градиенты © 2004 Александр Климов
'--------------------------------------
Private Sub Form1_Paint(ByVal sender As Object, ByVal e As
System.Windows.Forms.PaintEventArgs) Handles MyBase.Paint
        Dim g As Graphics = e.Graphics
        Dim rect As New Rectangle(10, 10, 90, 90)
        Dim p As New Pen(Color.RoyalBlue)
        g.DrawEllipse(p, rect)
        Dim lgb As New LinearGradientBrush(rect, _
           Color.RoyalBlue, Color.DeepSkyBlue, _
           LinearGradientMode.Horizontal)
        g.FillEllipse(lgb, rect)
End Sub
```

Сначала мы создаем перо и ограничивающий прямоугольник для создания круга. Затем с помощью знакомого нам метода `DrawEllipse` рисуем в заданной точке круг и заполняем его градиентом. В данном примере мы использовали горизонтальный градиент. В классе `LinearGradientBrush` существует еще несколько стилей градиента:

- `BackwardDiagonal` — градиент, идущий по диагонали от верхнего правого угла в нижний левый угол;
- `ForwardDiagonal` — градиент, идущий по диагонали от верхнего левого угла до нижнего правого угла;
- `Horizontal` — градиент, идущий слева направо;
- `Vertical` — градиент, идущий сверху вниз.

Градиенты можно применять не только к графическим объектам вроде эллипсов и прямоугольников, но и к текстам. Это естественно, так как шрифты фактически являются тоже графическими объектами, к которым применимы многие графические методы. Добавим код (листинг 5.2) к предыдущему примеру.

Листинг 5.2. Создание градиентного текста

```
        ' Градиентный текст
        Dim rec As New RectangleF(50, 80, 300, 90)
        Dim lgbText As New LinearGradientBrush(rec, _
           Color.Red, Color.Yellow, LinearGradientMode.ForwardDiagonal)
```

```
        Dim f As Font
        f = New Font("Tahoma", 36, Font.Style.Bold, GraphicsUnit.Pixel)
        g.DrawString("Visual Basic. NET", f, lgbText, 10, 110)
```

Данный код выводит под эллипсом текст, закрашенный градиентной кистью. Можно оживить данный пример. По умолчанию градиентное изменение цвета идет перпендикулярно указанной линии. Одна из версий конструктора позволяет изменять угол направления градиента. Можно воспользоваться данным обстоятельством для динамического изменения угла через таймер.В этом случае мы добьемся эффекта своеобразного вращающегося освещения. Подобный эффект мы можем наблюдать, например, на дискотеке (листинг 5.3).

Листинг 5.3. Анимационный эффект с градиентом

```
Private Sub Timer1_Tick(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Timer1.Tick
        ' Угол градиента
        Static angle As Integer

        Dim g As Graphics
        g = CreateGraphics()
        Dim rect As New Rectangle(150, 10, 90, 90)
        Dim p As New Pen(Color.RoyalBlue, 1)

        g.DrawEllipse(p, rect)

        Dim lbr As New LinearGradientBrush(rect, _
           Color.RoyalBlue, Color.DeepSkyBlue, angle)
        g.FillEllipse(lbr, rect)

        angle += 10
        If angle >= 360 Then
            angle = 0
        End If
        g.Dispose()
End Sub
```

В этом примере мы объявляем переменную `angle`, в которой будет содержаться угол направления градиента. Через заданный промежуток времени

мы меняем величину угла на 10 градусов, создавая эффект вращающегося освещения. К сожалению, бумага не способна передать создаваемый эффект, поэтому я не привожу на страницах книги рисунок описываемого примера. Придется вам запустить проект на своем компьютере, чтобы полюбоваться получаемой картинкой.

5.2. Замена "фотошопу"

Однажды в Интернете я нашел пример создания кнопки для веб-странички с помощью известного графического редактора Adobe Photoshop. Вот как описывался сам процесс:

1. Создаем квадратное изображение. На примере — 80×80 (если вы рисуете для веб-странички, то лучше сделать кнопку поменьше). С помощью инструмента выделения выделяем окружность. Чтобы выделить ровный круг, нажмите и удерживайте клавишу <Shift>, когда выделяете область.
2. Выбираем в качестве фонового цвета темный, а в качестве основного — что-нибудь посветлее. Используя инструмент плавной заливки (**Градиент**), закрашиваем выделение. Угол закрашивания зависит от желаемого угла падения света. На примере — угол 45°, т. е. из левого верхнего в правый нижний. Чтобы провести градиент точно под 45°, удерживайте клавишу <Shift>.
3. Теперь выделяем внутри маленькую окружность внутри большой. (Если выделение большой окружности еще сохранилось, то можно сжать большую окружность, выбрав команду меню **Select | Modify | Contrast**). Далее заполняем маленькую окружность градиентом тех же цветов, но в обратную сторону (из правого нижнего в левый верхний).
4. Последний штрих — надпись на кнопке. Ее лучше сделать контрастным цветом. Также для получения наилучшего результата в опциях инструмента печати для пункта **Anti-Aliasing** должен быть установлен флажок.

Прочитав эти строки, я подумал, что совсем не обязательно прибегать к помощи графического редактора. Не правда ли, так и хочется выполнить описываемые операции с помощью Visual Basic .NET 2003? Результат моих опытов находится в проекте `Photoshop` (листинг 5.4).

Листинг 5.4. Рисование необычной кнопки

```
'---------------------------------------
' Замена "фотошопу" © 2004 Александр Климов
'---------------------------------------
Imports System.Drawing.Drawing2D
```

```
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
        Dim g As Graphics = CreateGraphics()
        Dim grBrush As LinearGradientBrush
        Dim rect As New Rectangle(20, 20, 70, 70)
        Dim startColor As Color = Color.LightBlue
        Dim endColor As Color = Color.Blue

        grBrush = New LinearGradientBrush(rect _
           startColor, endColor, LinearGradientMode.ForwardDiagonal)
        g.FillEllipse(grBrush, rect)
        grBrush = New LinearGradientBrush(rect, _
           endColor, startColor, LinearGradientMode.ForwardDiagonal)
        g.FillEllipse(grBrush, New Rectangle(30, 30, 50, 50))

        g.Dispose()
    End Sub
```

Итак, следуя приведенной инструкции, я создал круг с градиентной заливкой со стилем `ForwardDiagonal`. Внутри созданного круга я поместил меньший круг, поменяв местами начальный и конечные цвета градиента. В результате получилась необычная объемная кнопочка, напоминающая пуговицу (рис. 5.1). Если у вас есть руководство по созданию различных эффектов в графических редакторах, попробуйте решить эту задачу в своих проектах. Это доставит вам массу удовольствия.

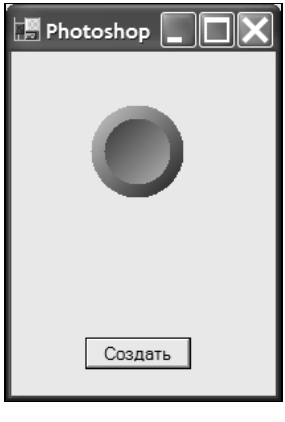

Рис. 5.1. Создание необычной кнопки

5.3. Класс *PathGradientBrush*

Класс `PathGradientBrush` аналогичен классу `LinearGradientBrush`, только заполнение фигуры идет не вдоль прямой, а на основе заданного пути. Использование данного класса дает очень интересные результаты, создавая иллюзии объемности — проект `PathGradientBrushSample` (листинг 5.5).

Листинг 5.5. Создание иллюзии объемности

```
'--------------------------------------
' Градиент-2 © 2004 Александр Климов
'--------------------------------------


    Private Sub Form1_Paint(ByVal sender As Object, ByVal e As
System.Windows.Forms.PaintEventArgs) Handles MyBase.Paint
        Dim wid As Integer = Me.ClientSize.Width
        Dim hgt As Integer = Me.ClientSize.Height
        Dim pts() As Point = { _
            New Point(wid * 0.5, hgt * 0.1), _
            New Point(wid * 0.7, hgt * 0.3), _
            New Point(wid * 0.9, hgt * 0.5), _
            New Point(wid * 0.7, hgt * 0.7), _
            New Point(wid * 0.5, hgt * 0.9), _
            New Point(wid * 0.3, hgt * 0.7), _
            New Point(wid * 0.1, hgt * 0.5), _
            New Point(wid * 0.3, hgt * 0.3)}

        Dim g_path As New GraphicsPath
        g_path.AddClosedCurve(pts, 0.5)

        ' Задаем кисть.
        Dim g_brush As New PathGradientBrush(g_path)
        Dim surround_colors() As Color = {Color.Pink}
        g_brush.SurroundColors = surround_colors
        g_brush.CenterColor = Color.White
        g_brush.CenterPoint = New PointF(wid * 0.4, hgt * 0.4)

        ' Заполняем фигуру.
        e.Graphics.FillPath(g_brush, g_path)
```

```
        ' Обрамляем фигуры.
        e.Graphics.DrawPath(Pens.Plum, g_path)

    End Sub
```

В этом примере я попытался изобразить что-то вроде подушки. Из примера (рис. 5.2) видно, что градиентная заливка формируется из массива точек, который затем передается методу `PathGradientBrush`.

Рис. 5.2. Создание объемности с помощью градиента

5.4. Шарики, мячики, пузыри

Пример создания объемной подушки получился несколько абстрактным. Вряд ли в своих программах вам придется создавать подобные предметы. Но саму возможность создания эффекта объемности с помощью класса `PathGradientBrush` несомненно надо взять на вооружение. Самое распространенное использование этого класса — создание выпуклых окружностей, имитирующих мячики, шарики, пузыри и другие подобные предметы — проект `Balls` (листинг 5.6).

Листинг 5.6. Создание шариков

```
'--------------------------------------
' Воздушные шарики © 2004 Александр Климов
'--------------------------------------
Imports System.Drawing.Drawing2D
```

```
Private Sub Form1_Paint(ByVal sender As Object, _
        ByVal e As System.Windows.Forms.PaintEventArgs) _
        Handles MyBase.Paint

    ' Создаем первый путь
    Dim gp As New GraphicsPath

    ' Рисуем первый шарик
    gp.AddEllipse(10, 10, 100, 100)

    ' Создаем градиентную кисть
    Dim pgBrush As New PathGradientBrush(gp)

    ' Задаем цвета для центра шарика
    pgBrush.CenterPoint = New PointF(70, 25)
    pgBrush.CenterColor = Color.SeaShell

    ' Цвет самого шарика
    Dim clr() As Color = {Color.FromArgb(255, 255, 0, 0)}
    pgBrush.SurroundColors = clr

    ' Выводим результат на экран
    e.Graphics.FillPath(pgBrush, gp)

    ' Повторяем все шаги для второго шарика
    Dim gp2 As New GraphicsPath
    gp2.AddEllipse(120, 10, 100, 150)
    Dim pgBrush2 As New PathGradientBrush(gp2)
    pgBrush2.CenterPoint = New PointF(180, 45)
    pgBrush2.CenterColor = Color.SeaShell
    Dim clr2() As Color = {Color.FromArgb(255, 50, 155, 100)}
    pgBrush2.SurroundColors = clr2
    e.Graphics.FillPath(pgBrush2, gp2)

    ' Веревочки для шариков
    e.Graphics.DrawLine(Pens.Black, 60, 110, _
```

```
                    CInt(ClientSize.Width / 2 - 1), _
                    ClientSize.Height - 1)
        e.Graphics.DrawLine(Pens.Black, 170, 160, _
                    CInt(ClientSize.Width / 2 - 1), _
                    ClientSize.Height - 1)

    End Sub
```

Главное, на что следует обратить внимание, — это определить центральную точку заливки. Сместив эту точку в сторону от центра эллипса, мы создаем на шарике эффект блика света (рис. 5.3).

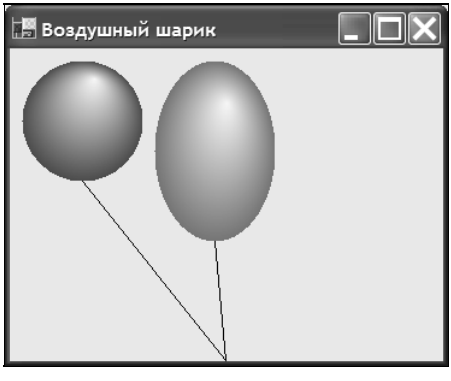

Рис. 5.3. Воздушные шарики

5.5. Мозаичный градиент

Работа с градиентами — очень увлекательное занятие. Я советую самостоятельно изучить документацию по этому вопросу. А напоследок расскажу еще об одном применении градиента при помощи мозаичного расположения кисти, почерпнутого из книги Ч. Петцольда "Программирование для Windows на Visual Basic .NET". Настоятельно рекомендую почитать эту книгу, где автор приводит несколько красивых примеров использования указанной техники. Особенно мне понравился пример с шестиугольниками. Вашему вниманию я хочу предложить небольшую модификацию этого примера, условно названную `Пчелиные соты` — проект `Honeycomb` (листинг 5.7).

Листинг 5.7. Создание пчелиных сот

```
'--------------------------------------
' Пчелиные соты © 2004 Александр Климов
'--------------------------------------
Imports System.Drawing.Drawing2D

    ' Размер одной соты (сторона и радиус)
    Const SizeSota As Single = 30

    Private Sub Form1_Paint(ByVal sender As Object, _
           ByVal e As System.Windows.Forms.PaintEventArgs) _
           Handles MyBase.Paint

        ' Вычисляем половину высоты шестиугольника
        Dim HalfSota As Single = CSng(SizeSota * Math.Sin(Math.PI / 3))

        ' Точки для шестиугольника, включая дополнительную ширину
        Dim aptf() As PointF = {New PointF(SizeSota, 0), _
                                New PointF(SizeSota * 1.5, 0), _
                                New PointF(SizeSota, 0), _
                                New PointF(SizeSota / 2, -HalfSota), _
                                New PointF(-SizeSota / 2, -HalfSota), _
                                New PointF(-SizeSota, 0), _
                                New PointF(-SizeSota * 1.5, 0), _
                                New PointF(-SizeSota, 0), _
                                New PointF(-SizeSota / 2, HalfSota), _
                                New PointF(SizeSota / 2, HalfSota)}

        ' Размеры формы
        Dim cx As Integer = ClientRectangle.Width - 1
        Dim cy As Integer = ClientRectangle.Height - 1

        ' Создаем первую кисть
        Dim pgBrush As PathGradientBrush = _
                    New PathGradientBrush(aptf, WrapMode.Tile)
        ' Цвет сот
        Dim clr() As Color = {Color.Yellow}
        pgBrush.SurroundColors = clr
```

```
        ' Сдвигаем соту и определяем вторую кисть
        Dim i As Integer
        For i = 0 To aptf.GetUpperBound(0)
            aptf(i).X += SizeSota * 1.5F
            aptf(i).Y += HalfSota
        Next i

        Dim pgBrush2 As PathGradientBrush = _
                            New PathGradientBrush(aptf, WrapMode.Tile)
        pgBrush2.SurroundColors = clr

        ' Выводим соты на экран
        Dim g As Graphics = e.Graphics
        g.FillRectangle(pgBrush, 0, 0, cx, cy)
        g.FillRectangle(pgBrush2, 0, 0, cx, cy)

    End Sub

    Private Sub Form1_Resize(ByVal sender As Object, ByVal e As Sys-
tem.EventArgs) Handles MyBase.Resize
        Refresh()
    End Sub
```

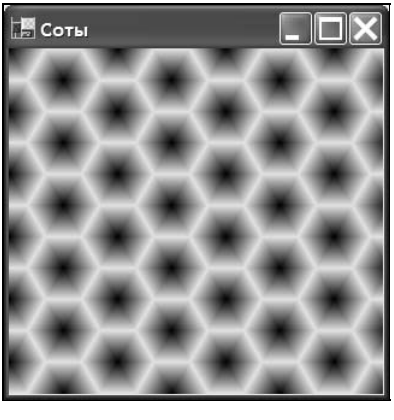

Рис. 5.4. Пчелиные соты

На мой взгляд, узор (рис. 5.4) получается просто восхитительным и просится в качестве обоев рабочего стола. Поиграйте с различными значениями цветов для кисти и размерами сот.

5.6. Советы и рекомендации

Градиентная заливка — весьма интересная тема для творчества. Используя самые различные сочетания цветов, стилей и фигур, можно получить очень красивые картинки, которые годятся для применения в качестве фона для рабочего стола или для ваших домашних страничек в Интернете. Постарайтесь самостоятельно придумать собственные узоры с использованием градиентных кистей.

Глава 6

Преобразования

Еще одно из преимуществ использования VB .NET вместо Visual Basic 6.0 это мощная поддержка преобразований. Одной строчкой кода можно менять визуальное представление графики на экране. Возможности настолько безграничны, что я просто теряюсь с выбором примеров. Начнем с простого.

6.1. Глобальные преобразования

Глобальное преобразование (world transform) позволяет трансформировать текст или графику самыми разными способами. Глобальные преобразования предполагают следующие правила:

- прямые при глобальных преобразованиях всегда остаются прямыми;
- параллельные прямые всегда остаются параллельными;
- объекты с равными размерами остаются объектами с равными размерами.

Перейдем к примерам. Запустите новый проект `WorldTransform`. В обработчике события `Form_Paint` пишем код (листинг 6.1).

Листинг 6.1. Глобальное преобразование текста

```
'--------------------------------------
' Глобальные преобразования © 2004 Александр Климов
'--------------------------------------
    Private Sub Form1_Paint(ByVal sender As Object, ByVal e As
System.Windows.Forms.PaintEventArgs) Handles MyBase.Paint
        Dim cx As Integer = ClientSize.Width
        Dim cy As Integer = ClientSize.Height
        Dim SomeText As String
        SomeText = "Тра-та-та, тра-та-та, мы везём с собой кота," _
           & vbNewLine & _
```

```
            "Чижика, собаку, петьку - забияку," _
            & vbNewLine & _
            "Обезьяну, попугая, вот - компания какая!" _
            & vbNewLine & _
            "Вот - компания какая!"
        Dim g As Graphics = e.Graphics

        g.DrawString(SomeText, Font, Brushes.Black, _
                        New RectangleF(0, 0, cx, cy))

    End Sub
```

Запустите пример. Вы увидите обычное окно формы с текстом песни С. Михалкова "Верные друзья". Ничего необычного. Теперь перед выводом текста при помощи метода `DrawString` добавьте строчку:

```
' поворачиваем текст на 30 градусов
g.RotateTransform(30)
```

Запустите проект снова. Всего одна строчка кода, а текст оказался повернут на 30 градусов (рис. 6.1). Ветераны бейсика подтвердят, что подобного эффекта в старой версии можно было добиться только с помощью системных функций Windows API. К слову сказать, вызовы методов `RotateTransform` носят кумулятивный характер. Если вы будете вызывать подряд этот метод с разными значениями углов, то поворот будет равен сумме заданных углов.

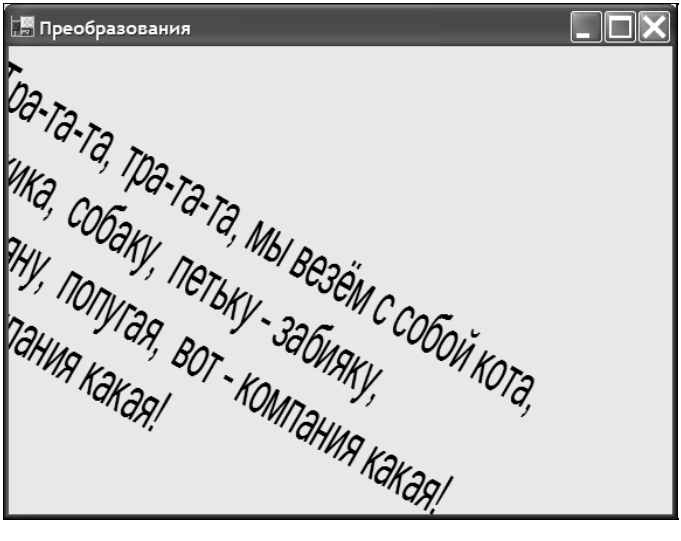

Рис. 6.1. Поворот текста на 30 градусов

Кроме рассмотренного, существуют и другие интересные методы. Например, попробуйте теперь добавить метод `ScaleTransform` перед вызовом метода `DrawString`:

```
' Изменяем масштабирование символов
g.ScaleTransform(2, 4)
```

В результате у нас увеличится высота символов текста вдвое, и в четыре раза увеличится их ширина. Существует также метод `TranslateTransform`, который сдвигает координаты выводимых символов по осям координат:

```
g.TranslateTransform(50, 10)
```

Интересные эффекты получаются, если комбинировать указанные методы. Обратите внимание, что можно использовать и отрицательные величины. В этом случае текст будет отображаться зеркально:

```
g.TranslateTransform(cx / 2, cy / 2)
g.ScaleTransform(1, -1)
g.DrawString(SomeText, Font, Brushes.Black, New RectangleF(0, 0, cx, cy))
```

Сначала мы устанавливаем координаты начала вывода текста в центре формы. Затем, используя отрицательное значение, отражаем текст зеркальным образом. Попробуйте также использовать другие комбинации:

```
g.ScaleTransform(-1, 1)
```

или:

```
g.ScaleTransform(-1, -1)
```

6.2. Матрица

Преобразования возможны и с помощью класса `Matrix`. Всем поклонникам фильма братьев Вачовски "Матрица" использовать этот класс в своих проектах обязательно! Впрочем, если без шуток, это действительно очень интересный класс, с помощью которого можно добиться многих интересных эффектов. Для первого знакомства с классом приведем простой пример — проект `Matrix` (листинг 6.2).

Листинг 6.2. Использование матрицы

```
'--------------------------------------
' Матрица © 2004 Александр Климов
'--------------------------------------
    Private Sub Form1_Paint(ByVal sender As Object, _
            ByVal e As System.Windows.Forms.PaintEventArgs) _
            Handles MyBase.Paint
```

```
        Dim myPen As New Pen(Color.Blue, 1)
        Dim myPen2 As New Pen(Color.Red, 1)
        Dim rotatePoint As New PointF(150.0F, 50.0F)

        ' Рисуем первый прямоугольник синего цвета
        e.Graphics.DrawRectangle(myPen, 150, 50, 180, 100)

        ' Создаем матрицу и поворачиваем на 30 градусов
        Dim myMatrix As New System.Drawing.Drawing2D.Matrix
        myMatrix.RotateAt(30, rotatePoint, MatrixOrder.Append)

        ' Рисуем второй прямоугольник красного цвета
        ' после использования матрицы
        e.Graphics.Transform = myMatrix
        e.Graphics.DrawRectangle(myPen2, 150, 50, 180, 100)
    End Sub
```

Суть примера очень проста. Сначала мы рисуем первый прямоугольник синего цвета. Затем создаем объект типа `Matrix` и используем его метод `RotateAt` для поворота на 30 градусов. Передаем информацию о трансформации объекту `Graphics` и рисуем прямоугольник красного цвета с такими же размерами. Вы увидите, что в результате преобразования наш первый прямоугольник повернулся на 30 градусов вокруг своей левой верхней вершины. Естественно, матрицу можно применять и к тексту (листинг 6.3).

Листинг 6.3. Использование матрицы в текстах

```
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click

        Static angle As Long    ' поворот в градусах

        ' Создаем матрицу для поворота вокруг точки 70,70
        Dim rotation_matrix As New System.Drawing.Drawing2D.Matrix
        angle += 20
        rotation_matrix.RotateAt(angle, New PointF(70, 70))

        ' Создаем объект Graphics и преобразуем его
        Dim g = Me.CreateGraphics()
        g.Transform = rotation_matrix
```

```
        g.clear(Color.Black)
        g.DrawString("Матрица", Font, Brushes.Green, _
                     New RectangleF(70, 70, 150, 150))
    End Sub
```

В данном случае я вынес код в событие кнопки. Теперь при каждом нажатии на кнопку заданный текст будет поворачиваться на 20 градусов по часовой стрелке (рис. 6.2).

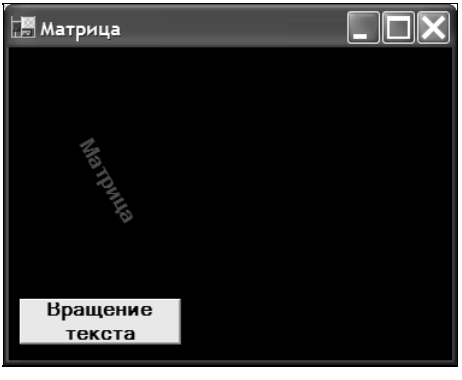

Рис. 6.2. Вращение текста с использованием матрицы

6.3. Вращение линии

Используя данные методы, можно легко заставить вращаться простую линию с помощью таймера — проект `RotatedLine` (листинг 6.4).

Листинг 6.4. Вращение линии вокруг ее центра

```
'--------------------------------------
' Вращение линии © 2004 Александр Климов
'--------------------------------------
Imports System.Drawing.Drawing2D
    ' Глобальная переменая для вращения
    Public f As Integer

    Private Sub Form1_Paint(ByVal sender As Object, ByVal e As
System.Windows.Forms.PaintEventArgs) Handles MyBase.Paint
```

```
        Dim g As Graphics = e.Graphics
        ' Вспомогательная окружность, внутри которой вращается линия
        'g.DrawEllipse(New Pen(Color.Violet, 3), 30, 30, 150, 150)

        Dim gp As New GraphicsPath
        gp.AddLine(105, 30, 105, 180)

        Dim rotationTransform As New Matrix(1, 0, 0, 1, 1, 1)
        Dim rotationPoint = New PointF(105.0F, 105.0F)

        rotationTransform.RotateAt(f, rotationPoint)
        gp.Transform(rotationTransform)

        g.DrawPath(New Pen(Color.BlueViolet, 3), gp)

        f = f + 10
        If f = 360 Then
            f = 0
        End If
    End Sub

    Private Sub Timer1_Tick(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Timer1.Tick
        Invalidate()
    End Sub
```

В этом примере отрезок прямой вращается внутри некоторой окружности. Если хотите увидеть эту окружность, то снимите знак комментария со строки

```
'g.DrawEllipse(New Pen(Color.Violet, 3), 30, 30, 150, 150)
```

Уменьшив длину линии в два раза, мы получим простенький образец стрелки часов. О создании часов поговорим отдельно.

6.3.1. Часики

С помощью преобразований можно написать и более сложные примеры. Когда я писал книгу, по телевизору звучала песня Валерии "Часики". Взглянув на свои настенные часы, я увидел вращающуюся секундную стрелку и подумал, почему бы нам не написать пример, рисующий настенные аналоговые часы? Запускаем новый проект `AnalogClock`. Так как у часов имеются

три стрелки, выполняющие одинаковую работу — вращение по кругу, то для рисования стрелок я выделил отдельный класс `cHands` (листинг 6.5).

Листинг 6.5. Класс для стрелок часов

```
'--------------------------------------
' Аналоговые часы © 2004 Александр Климов
'--------------------------------------

Imports System.Drawing.Drawing2D

' Класс для создания стрелок
Public Class cHands
    Private angle As Integer = 0
    Public gp As New GraphicsPath
    Public bfilled As Boolean = False

    ' Процедура расчета углов поворота
    Public Sub SetAngle(ByVal newAngle As Integer, ByVal imgCenter As
PointF)
        Dim rotate As Integer = newAngle - Angle
        Dim trans As New Matrix
        trans.RotateAt(rotate, imgCenter)
        gp.Transform(trans)
        Me.Angle = newAngle
    End Sub

    ' Процедура рисования
    Public Sub DrawClock(ByVal imgDraw As Graphics)
        If bfilled Then
            imgDraw.FillPath(Brushes.DeepPink, gp)
            imgDraw.DrawPath(Pens.Yellow, gp)
        Else
            imgDraw.DrawPath(New Pen(Color.DimGray, 1), gp)
        End If
    End Sub
End Class
```

Сначала я объявил в проекте новый класс `cHands`. И сразу же в первых строчках создал три переменные, определяющие угол поворота стрелок (`angle`), графический объект (`gp`) и булеву переменную для заливки (`bfilled`). Процедура `SetAngle` просто вычитает величину нового угла из прежнего угла и поворачивает стрелку на полученную величину с помощью метода `RotateAt`. Процедура `DrawClock` обрамляет часовую и минутную стрелки для лучшего восприятия и заполняет их заданным цветом. Для секундной стрелки можно просто ограничиться кистью с толщиной в один пиксел. Используя данный класс, мы сможем достаточно просто нарисовать стрелки и придать им нужные свойства. Понятно, что нам не обойтись в нашем проекте без таймера. Добавляем в проекте на основную форму таймер `Timer1` и сразу присваиваем его свойству `Enabled` значение `True`. Наши часы будут рисоваться не на самой форме, а на графическом поле. Добавляем в проект графическое поле `PictureBox`. Присваиваем ему имя `picClock`. На этом визуальное программирование закончено. Пора снова переходить в редактор кода (листинг 6.6).

Листинг 6.6. Создание часов

```
Imports System.Drawing.Drawing2D

    ' Переменные, отвечающие за вывод стрелок и центра часов
    Private hHour As New cHands
    Private hMinute As New cHands
    Private hSecond As New cHands
    Private center = New PointF(150, 150)

    Private Sub initclock()

        ' Размеры графического поля должны быть 300 x 300
        Me.ClientSize = New Size(300, 300)

        ' Массив для нескольких точек. Каждая точка соединяется с другой.
        ' В результате получается форма для часовой стрелки.
        Dim drawHour As Point() = { _
          New Point(Int(picClock.Width / 2), Int(picClock.Height / 2)), _
          New Point(Int(picClock.Width / 2 - 5), 80), _
          New Point(Int(picClock.Width / 2), 50), _
          New Point(Int(picClock.Width / 2 + 5), 80), _
          New Point(Int(picClock.Width / 2), Int(picClock.Height / 2))}
```

```
        ' Добавляем эти точки к объекту с помощью свойства AddLines
        ' объекта GraphicsPath
        hHour.gp.AddLines(drawHour)
        ' Заполняем стрелку
        hHour.bfilled = True

        ' Аналогично поступаем с минутной стрелкой
        Dim drawMinute As Point() = { _
                            New Point(Int(picClock.Width / 2), _
                            Int(picClock.Height / 2)), _
                            New Point(Int(picClock.Width / 2 - 5), 60), _
                            New Point(Int(picClock.Width / 2), 30), _
                            New Point(Int(picClock.Width / 2 + 5), 60), _
                            New Point(Int(picClock.Width / 2), _
                            Int(picClock.Height / 2))}

        hMinute.gp.AddLines(drawMinute)
        hMinute.bfilled = True

        ' Секундная стрелка. Не заполняем ее
        hSecond.gp.AddLine(center, _
           New PointF(Int(picClock.Width / 2), 20))
    End Sub

    Private Sub Timer1_Tick(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Timer1.Tick
        picClock.Invalidate()
    End Sub

    Private Sub picClock_Paint(ByVal sender As Object, ByVal e As
System.Windows.Forms.PaintEventArgs) Handles picClock.Paint
        DrawFace(e.Graphics)
        hHour.setAngle((360 / 12) * Val(Format(Now, "hh")), center)

        hMinute.SetAngle((360 / 60) * Val(Format(Now, "mm")), center)
        hSecond.SetAngle((360 / 60) * Val(Format(Now, "ss")), center)

        hHour.DrawClock(e.Graphics)
        hMinute.DrawClock(e.Graphics)
```

```
        hSecond.DrawClock(e.Graphics)
    End Sub

    Private Sub DrawFace(ByVal g As Graphics)
        Dim path As String = _
           Mid(Application.ExecutablePath, 1, _
           InStrRev(Application.ExecutablePath, "\"))
        Dim bBoundClock As New HatchBrush(_
          HatchStyle.LargeConfetti, Color.Green, Color.White)

       g.FillEllipse(bBoundClock, New Rectangle(5, 5, 290, 290))
       g.FillEllipse(Brushes.WhiteSmoke, New Rectangle(15, 15, 270, 270))
       g.FillEllipse(Brushes.Green, New Rectangle(140, 140, 20, 20))

        Dim f As New Font("Arial", 10, FontStyle.Bold)
        Dim stringSize = g.MeasureString("BHV", f)
        g.DrawString("BHV", f, Brushes.BlueViolet, _
                    (300 - stringSize.width) / 2, 200)

        bBoundClock.Dispose()
        ' Не удаляйте объект g!
    End Sub
End Class
```

Чтобы не мучиться с вычислениями, я жестко задал размеры окна программы (300×300 пикселов), и центр формы, таким образом, находится в точке с координатами (150, 150). Часовая и минутная стрелки рисуются с помощью массива точек, которые добавляются к линии с помощью метода `AddLines` и заполняются цветом. Секундная стрелка не требует подобного подхода из-за своей толщины. Достаточно одной простой линии. Обратите внимание, что формула вычисления угла для часовой стрелки отличается от формул для минутной и секундной стрелки. Надеюсь, вы знаете, что на циферблатах часов используются 12 делений для часов и 60 делений для секунд и минут (если не верите, то не поленитесь — подсчитайте вручную). Поэтому мы делим окружность на 12 или 60 равных частей и умножаем на текущее время. Полученный результат передаем процедуре `SetAngle`, которая вычисляет необходимый угол, и выводим стрелки в нужном положении при помощи процедуры `DrawClock`.

Для контура часов я использовал узорную кисть `HatchBrush` со стилем `LargeConfetti`. Три вызова метода `FillEllipse` рисуют три круга — контур

часов с узором, циферблат цвета белого дыма и центральную ось зеленого цвета. Для большего сходства я добавил в нижней части циферблата небольшую надпись производителя часов. Мне пока ничего не известно о планах издательства "БХВ-Петербург" по выпуску настенных часов, но надеюсь, они меня не забудут и поделятся частью своей прибыли. Основная часть программы выполнена. Можете запустить проект и посмотреть на работающие часы (рис. 6.3). Вы должны увидеть, что секундная стрелка вращается, а часовая и минутная стрелки часов стоят в нужных местах (проверьте по вашим компьютерным часам в нижнем правом углу экрана).

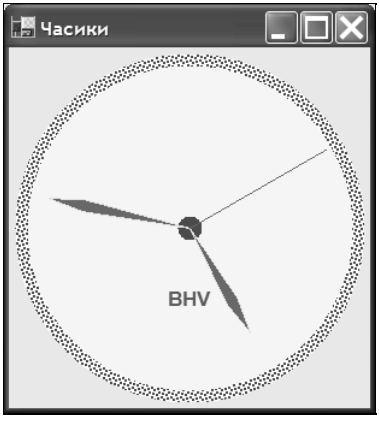

Рис. 6.3. Часы

Кстати, в процессе написания кода я допустил ошибку из-за невнимательности и поместил вызов процедуры `DrawFace` самым последним в указанном событии. В результате после запуска программы я не увидел никаких стрелок. К счастью, проект не самый сложный, а также из-за моей дурной привычки запускать проект после каждой новой строчки кода, мне удалось достаточно быстро понять причину пропажи стрелок. Но в более сложных программах такая невнимательность может обернуться потерей драгоценного времени в поисках ошибки. Особенно часто такая ситуация возникает при изучении чужих программ, которые мне иногда присылают. Авторы присланных программ часто недоумевают, почему программа не работает должным образом, и просят найти ошибку. А источником подобных ошибок частенько бывает элементарная забывчивость и невнимательность.

Примечание

Данный абзац не следует воспринимать слишком буквально. Не стоит закидывать меня письмами со своими неработающими примерами. Девяносто девять процентов вероятности, что ваше письмо уйдет в корзину для мусора...

Но вернемся к нашим часам. Я предложил вам всего лишь приблизительный алгоритм создания аналоговых часов. Предела совершенству нет. Например, стрелки часов можно сделать весьма причудливой формы, используя кривые Безье. Дайте волю своей фантазии.

6.4. Сдвиг

Еще одним интересным видом преобразований являются *сдвиги*. В проекте `Shear` я покажу два простых примера горизонтального и вертикального сдвига (листинг 6.7). Эффект сдвига достигается путем манипулирования значений в классе `Matrix`.

Листинг 6.7. Создание сдвигов

```
'------------------------------
' Сдвиг © 2004 Александр Климов
'------------------------------
Imports System.Drawing.Drawing2D


    Private Sub Form1_Paint(ByVal sender As Object, ByVal e As
System.Windows.Forms.PaintEventArgs) Handles MyBase.Paint
        Dim myMatrix As New Matrix
        Dim f As New Font("Tahoma", 18, FontStyle.Bold)

        myMatrix = New Matrix(1, 0, -0.5, 1, 0, 0)
        e.Graphics.Transform = myMatrix
        e.Graphics.DrawString(_
          "Не тяни кота за хвост", f, Brushes.Blue, 60, 30)

        e.Graphics.ResetTransform()
        myMatrix = New Matrix(1, 0, 0.5, 1, 0, 0)
        e.Graphics.Transform = myMatrix
        e.Graphics.DrawString(_
          "Не тяни кота за хвост", f, Brushes.Blue, 10, 70)

        ' Рисуем прямоугольник красного цвета
        e.Graphics.ResetTransform()
        e.Graphics.Transform = New Matrix(1, 0.5, 0, 1, 0, 1)
        e.Graphics.DrawRectangle(Pens.Red, 80, 80, 160, 100)
    End Sub
```

Если запустить проект, то мы увидим на экране две строчки, наклоненные в разные стороны. Это пример горизонтального сдвига. Все буковки наклонены в одну из сторон, но при этом находятся на одной линии. Красный параллелограмм на самом деле является примером вертикального сдвига стандартного прямоугольника (рис. 6.4).

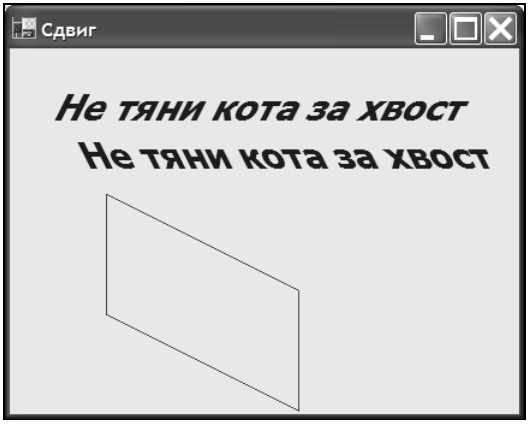

Рис. 6.4. Горизонтальный и вертикальный сдвиги

6.5. Советы и рекомендации

Преобразования — это новое слово для программистов на Visual Basic, и оно таит в себе очень большие возможности. Признаюсь, я сам еще глубоко в них не вникал. Кстати сказать, совсем не обязательно использовать только горизонтальный или только вертикальный сдвиг. Можно объединить эти преобразования и получить сдвиг по двум направлениям.

Глава 7

Фигуры

В нашем мире существуют различные фигуры, знакомые нам с детства. Попытаемся воспроизвести их с помощью Visual Basic .NET 2003.

7.1. Рисуем звезду

Пятиконечная звезда встречается нам на каждом шагу. Когда-то звезда была элементом герба Советского Союза, да и сейчас входит в состав гербов многих стран. Но на самом деле пятиконечная звезда (пентаграмма) была известна еще в далекой древности и считалась у некоторых народов амулетом здоровья.

7.1.1. Звезда — первый способ

Давайте попробуем нарисовать звезду самостоятельно — проект `Stars` (листинг 7.1).

Листинг 7.1. Создание звезды

```
'--------------------------------------
' Звезды © 2004 Александр Климов
'--------------------------------------
Imports System.Drawing.Drawing2D

    Private Sub butStar1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles butStar1.Click

        Dim cx As Integer = 150
        Dim cy As Integer = 150
        Dim apt(4) As Point
```

```
        Dim i As Integer
        For i = 0 To apt.GetUpperBound(0)
            Dim theta As Double = (i * 1.2 - 0.5) * Math.PI
            apt(i).X = CInt(cx * (1 + 0.7 * Math.Cos(theta)))
            apt(i).Y = CInt(cy * (1 + 0.7 * Math.Sin(theta)))
        Next i

        Dim g As Graphics = CreateGraphics()
        Dim br As New SolidBrush(Color.Red)
        g.DrawLine(New Pen(Color.Red, 3), apt(0), apt(1))
        g.DrawLine(New Pen(Color.Blue, 3), apt(1), apt(2))
        g.DrawLine(New Pen(Color.Blue, 3), apt(2), apt(3))
        g.DrawLine(New Pen(Color.Blue, 3), apt(3), apt(4))
        g.DrawLine(New Pen(Color.Green, 3), apt(0), apt(4))
        g.FillPolygon(br, apt, FillMode.Alternate)
        g.Dispose()
    End Sub
```

Запустите программу, и вы увидите красную звезду с закрашенными вершинами (рис. 7.1). Разберем поподробнее код. Сначала мы объявляем две переменные `cx` и `cy`, которые характеризуют смещение начальной точки рисования относительно начала координат. Далее идет цикл построения звезды с помощью пяти линий. Здесь надо остановиться поподробнее:

```
Dim theta As Double = (i * 1.2 - 0.5) * Math.PI
```

Сначала нам надо задать угол и его смещение в радианах при каждом шаге цикла. Чтобы звезда стояла ровно, сместим первую точку на 90 градусов:

`0.5 * Math.PI` — в радианах это 90 градусов (начальный угол смещения).

Величина 0,7 является размером вершин звезды. Можете изменять этот параметр для увеличения или уменьшения фигуры. `Math.Cos(theta)` — поворот вектора.

```
apt(i).X = CInt(cx * (1 + 0.7 * Math.Cos(theta)))
apt(i).Y = CInt(cy * (1 + 0.7 * Math.Sin(theta)))
```

Далее мы поочередно рисуем линии, причем первую и последнюю линии — разными цветами для большей наглядности. Если хотите частично закрасить звезду, то используйте метод `FillMode`:

```
g.FillPolygon(br, apt, FillMode.Alternate)
```

Естественно, в этом случае вызов методов `DrawLine` можно опустить. Для полной закраски звезды используйте параметр `Winding`:

```
g.FillPolygon(br, apt, System.Drawing.Drawing2D.FillMode.Winding)
```

Рис. 7.1. Рисуем звезду. Первый способ

7.1.2. Звезда — второй способ

Второй способ заключается в рисовании звезды по контуру. Размещаем на форме вторую кнопку и пишем следующий код (листинг 7.2).

Листинг 7.2. Рисование звезды другим способом

```
    Private Sub butStar2_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles butStar2.Click
        Dim g As Graphics = CreateGraphics()
        Dim cx As Integer = ClientRectangle.Width / 2
        Dim cy As Integer = ClientRectangle.Height / 2
        Dim outerR As Integer = 120
        Dim innerR As Integer = 46
        Dim pn As New Pen(Color.Red, 3)
        Dim pn2 As New Pen(Color.Blue, 3)
```

```
    Dim i As Integer
    For i = 0 To 10
        If i Mod 2 = 1 Then
            g.DrawLine(pn, _
 CInt(cx + Math.Ceiling(outerR * Math.Cos((i + 1) * Math.PI / 5))), _
 CInt(cy + Math.Ceiling(outerR * Math.Sin((i + 1) * Math.PI / 5))), _
 CInt(cx + Math.Ceiling(innerR * Math.Cos(i * Math.PI / 5))), _
 CInt(cy + Math.Ceiling(innerR * Math.Sin(i * Math.PI / 5))))

        Else
            g.DrawLine(pn2, _
 CInt(cx + Math.Ceiling(outerR * Math.Cos(i * Math.PI / 5))), _
 CInt(cy + Math.Ceiling(outerR * Math.Sin(i * Math.PI / 5))), _
 CInt(cx + Math.Ceiling(innerR * Math.Cos((i + 1) * Math.PI / 5))), _
 CInt(cy + Math.Ceiling(innerR * Math.Sin((i + 1) * Math.PI / 5))))

        End If
    Next

    g.Dispose()
End Sub
```

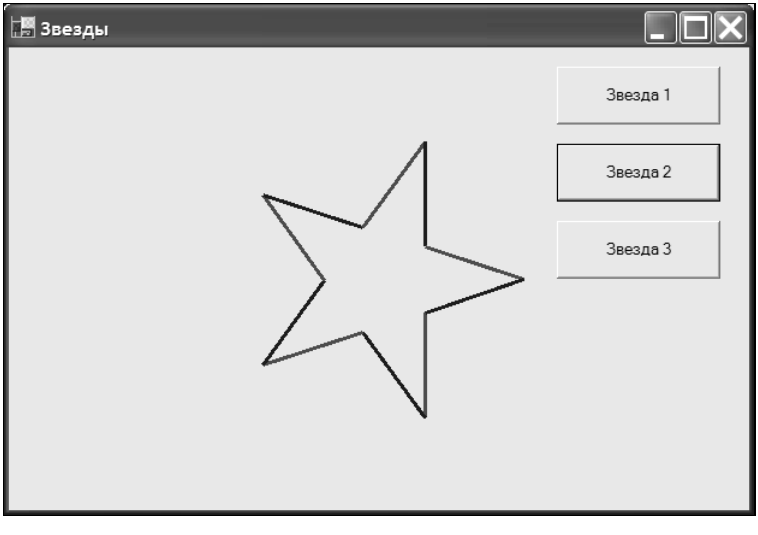

Рис. 7.2. Рисуем звезду по контуру

Первые три строчки, я думаю, вам понятны. Далее задаем внешний и внутренний радиусы окружностей, на которых будут лежать вершины и впадины звезды общим числом 10 точек. Так как вершины и впадины чередуются, то в цикле мы разделяем эти точки по признаку четности и соединяем их линиями (рис. 7.2).

7.1.3. Градиентная звезда

Однако вернемся к первой звезде. Немного изменив код, можно получить необычную звезду с градиентной заливкой (рис. 7.3). Основой этого способа послужил один из примеров книги Ч. Петцольда "Программирование для Windows на Visual Basic .NET" (листинг 7.3).

Листинг 7.3. Создание градиентной звезды

```
    Private Sub butStar3_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles butStar3.Click
        Dim cx As Integer = 150
        Dim cy As Integer = 150

        Dim apt(4) As Point

        Dim i As Integer
        For i = 0 To apt.GetUpperBound(0)
            Dim theta As Double = (i * 1.2 - 0.5) * Math.PI
            apt(i).X = CInt(cx * (1 + 0.7 * Math.Cos(theta)))
            apt(i).Y = CInt(cy * (1 + 0.7 * Math.Sin(theta)))
        Next i

        Dim pgBrush As New PathGradientBrush(apt)
        pgBrush.CenterColor = Color.SeaShell
        pgBrush.SurroundColors = New Color() {Color.Red}

        Refresh()

        Dim g As Graphics = CreateGraphics()
        g.FillPolygon(pgBrush, apt, FillMode.Winding)
        g.Dispose()
    End Sub
```

Рис. 7.3. Градиентная звезда

7.2. Фигура Инь-Янь

Фигура Инь-Янь — древний восточный символ. Инь — в древнекитайской мифологии является темным женским, а Янь — светлым мужским началом, которые почти всегда существуют вместе в постоянном взаимодействии. Особенностью этой фигуры является то, что она состоит из двух одинаковых частей, которые совместно образуют круг (рис. 7.4). Если рассматривать две половинки отдельно, то это трудно себе представить. Попробуем программно создать такую фигуру — проект `Ying-Yang` (листинг 7.4)

Листинг 7.4. Создание фигуры Инь-Янь

```
'--------------------------------------
' Фигура Инь-Янь © 2004 Александр Климов
'--------------------------------------

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
        ' Рисуем фигуру Инь-Янь
        Dim g As Graphics = CreateGraphics()
        Dim black_pen As SolidBrush = New SolidBrush(Color.Black)
        Dim white_pen As SolidBrush = New SolidBrush(Color.White)
```

```
        g.FillEllipse(black_pen, 100, 100, 200, 200)
        ' Делаем паузу в одну секунду
        System.Threading.Thread.Sleep(1000)
        g.FillEllipse(white_pen, 150, 100, 100, 100)
        System.Threading.Thread.Sleep(1000)
        g.FillPie(white_pen, 100, 100, 200, 200, -90, 180)
        System.Threading.Thread.Sleep(1000)
        g.FillEllipse(black_pen, 150, 200, 100, 100)
        System.Threading.Thread.Sleep(1000)
        g.FillEllipse(black_pen, 185, 135, 30, 30)
        g.FillEllipse(white_pen, 185, 235, 30, 30)
        g.Dispose()
    End Sub
```

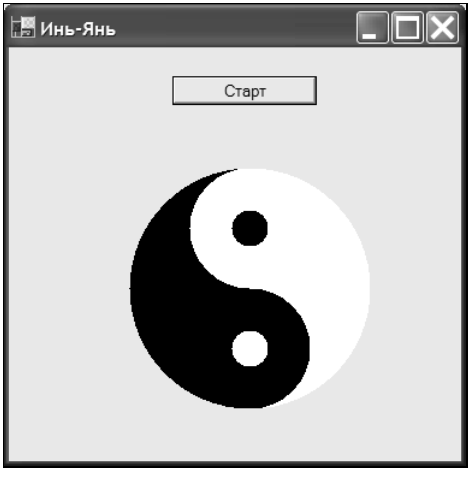

Рис. 7.4. Рисуем фигуру Инь-Янь

Как видно из кода, фигура получается путем наложения заполненных окружностей и одной полуокружности (метод `DrawPie`) белого и черного цветов друг на друга. Алгоритм получения фигуры прост. Сначала на форме рисуется большой заполненный черный круг. Затем круг условно делится на две равных части по горизонтали. В верхней части вписывается белый круг, диаметр которого вдвое меньше диаметра большого круга. Потом большой круг делится на две половинки по вертикали и одна из них закрашивается белым цветом. В нижней части круга рисуется черный круг, аналогичный белому. Последний этап — в центре двух малых кругов рисуются маленькие

кружочки противоположного цвета. Чтобы процесс создания был более наглядным, между вызовами методов я сделал паузу в одну секунду.

7.3. Фракталы

Фракталы — это особый вид узоров, завораживающих своей красотой и таинственностью, проявляется в самых неожиданных областях: метеорологии, философии, географии, биологии, механике. Если задать слово "фрактал" в любой поисковой системе, то найдется множество ссылок на красивые картинки, полученные с помощью фракталов. Самое простое определение: фрактал — это такой объект, для которого не важно, с каким увеличением его рассматривать в лупу, так как при любом увеличении структура его остается одной и той же. Большие по масштабу структуры полностью повторяют структуры, меньшие по масштабу.

Простейший пример, понятный многим, — миллиметровая бумага. Большой лист такой бумаги можно считать большим квадратом, который в свою очередь разделяется на более мелкие квадраты. Причем процесс можно повторять очень долго, пока сторона квадрата не станет равной одному миллиметру. Сеточка из более мелких квадратов не слишком интересна для нашего взора. Существуют и другие способы разбиения исходной фигуры, о которых будет сказано далее.

Автором термина считается Бенуа Мандельброт (Benoit Mandelbrot). Сам Мандельброт вывел слово *fractal* от латинского слова *fractus*, что означает разбитый (поделенный на части). И одно из определений фрактала — это геометрическая фигура, состоящая из частей, которая может быть поделена на части, каждая из которых будет представлять уменьшенную копию целого (по крайней мере, приблизительно). Так, в одном из примеров Мандельброт предлагает рассмотреть линию побережья с самолета, стоя на ногах и в увеличительное стекло. Во всех случаях получим одни и те же узоры, но только меньшего масштаба. Первые идеи фрактальной геометрии возникли у ученых еще в XIX веке. Георг Кантор с помощью простой рекурсивной (повторяющейся) процедуры превратил линию в набор несвязанных точек (так называемая *пыль Кантора*). Он брал линию, удалял центральную треть и после этого повторял то же самое с оставшимися отрезками. Джузеппе Пеано нарисовал особый вид линии при помощи следующего алгоритма. На первом шаге он брал прямую линию и заменял ее на 9 отрезков длиной в 3 раза меньшей, чем длина исходной линии. Далее он делал то же самое с каждым отрезком получившейся линии. И так до бесконечности. Уникальность *кривой Пеано* в том, что она заполняет всю плоскость. Доказано, что

для каждой точки на плоскости можно найти точку, принадлежащую линии Пеано. Кривая Пеано и пыль Кантора выходили за рамки обычных геометрических объектов. Они не имели четкой размерности. Пыль Кантора строилась вроде бы на основании одномерной прямой, но состояла из точек (размерность 0). А кривая Пеано строилась на основании одномерной линии, а в результате получалась плоскость. Вплоть до XX века шло накопление данных о таких странных объектах, без какой-либо попытки их систематизировать.

Так было, пока за них не взялся уже упоминавшийся нами Бенуа Р. Мандельброт, математик из Исследовательского центра им. Томаса Уотстона при IBM, который считается отцом современной фрактальной геометрии. Работая в IBM математическим аналитиком, он изучал шумы в электронных схемах, которые невозможно было описать с помощью статистики. Постепенно сопоставив факты, он пришел к открытию нового направления в математике — фрактальной геометрии. Почему же фракталы так красивы? Математика вся пронизана красотой и гармонией, только эту красоту надо увидеть. Вот как пишет сам Мандельброт в своей книге "The Fractal Geometry of Nature":

"Почему геометрию часто называют холодной и сухой? Одна из причин лежит в ее неспособности описать форму облаков, гор или деревьев. Облака — это не сферы, горы — не углы, линия побережья — не окружность, кора не гладкая, а молния не прямая линия..."

7.3.1. Программирование фракталов

Псевдокод, описывающий общий принцип фрактала:

```
Объект рисуется в достаточно большом масштабе
Части этого объекта заменяются меньшими копиями этого объекта
```

Ясно, что это описание рекурсивного процесса. Рассмотрим первый пример создания фрактала. Псевдокод выглядит так:

```
Процедура Рисования Квадрата
    У каждого угла квадрата рисуется квадрат меньшего размера
    Рисуется квадрат, если не существуют квадраты меньшего размера
Конец Процедуры
```

Так будет выглядеть код на VB .NET 2003 — проект `Fractals` (листинг 7.5).

Листинг 7.5. Рисование простейшего фрактала

```
'--------------------------------------
' Фрактал из квадратов © 2004 Александр Климов
'--------------------------------------
    Sub Square(ByRef x As Integer, ByRef y As Integer, ByRef Size As
Integer)

        ' для выхода из процедуры
        If Size < 20 Then Exit Sub

        Dim g As Graphics = CreateGraphics()
        g.DrawRectangle(Pens.Red, x, y, Size, Size)
        ' левый верхний угол
        Square(x - Size / 4, y - Size / 4, Size / 2)

        ' правый верхний угол
        Square(x + Size - Size / 4, y - Size / 4, Size / 2)

        ' левый нижний угол
        Square(x - Size / 4, y + 3 * Size / 4, Size / 2)

        ' правый нижний угол
        Square(x + Size - Size / 4, y + 3 * Size / 4, Size / 2)

        g.Dispose()
    End Sub

    Private Sub mnuSquare_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles mnuSquare.Click
        Refresh()
        Square(200, 200, 250)
    End Sub
```

При выполнении каждой рекурсии появляются четыре новых квадрата. Каждый из них составляет одну четвертую от предыдущего. Результат представлен на рис. 7.5.

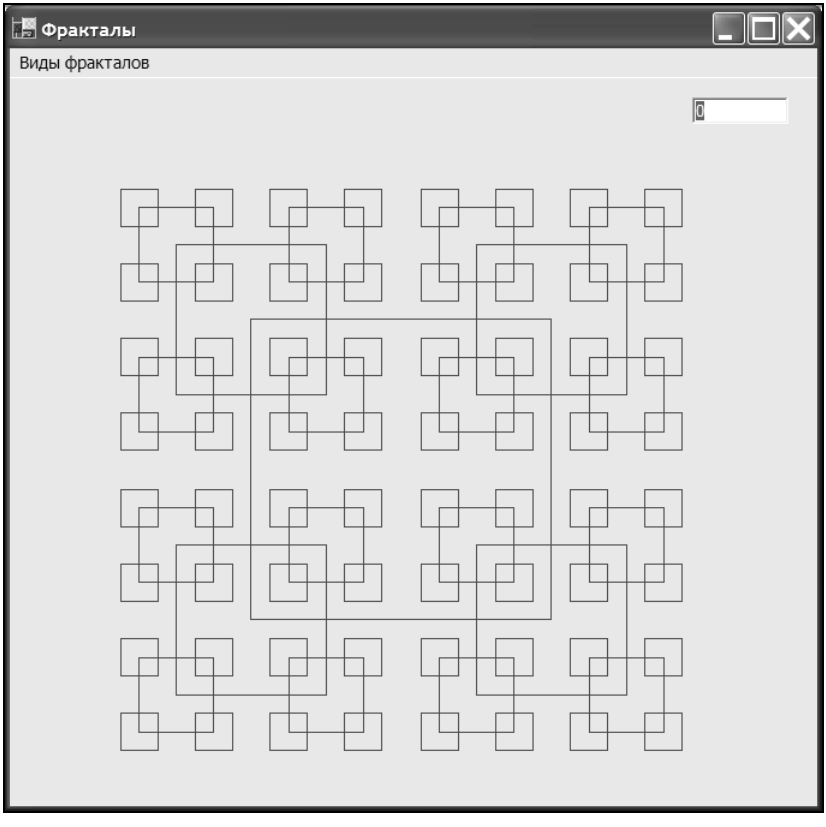

Рис. 7.5. Простейший фрактал из квадратов

7.3.2. Узорная скатерть

Теперь, когда вы поняли принцип построения фракталов, можете придумать свои правила. Очень интересный узор получается, если воображаемый квадрат соединить диагоналями и двумя отрезками, проходящими через середины сторон квадрата. Затем уменьшаем квадрат в два раза и размещаем сверху, снизу, слева и справа (листинг 7.6). В итоге мы получим своеобразную скатерть, сотканную из узоров (рис. 7.6).

Листинг 7.6. Создание узорной скатерти

```
'--------------------------------------
' Скатерть © 2004 Александр Климов
'--------------------------------------
```

```
    Sub Uzor(ByRef x As Integer, ByRef y As Integer, ByRef size As
Integer)
        ' для выхода из процедуры
        If size < 5 Then Exit Sub

        Dim g As Graphics = CreateGraphics()

        ' Диагонали
        g.DrawLine(Pens.Red, x, y, x + size, y + size)
        g.DrawLine(Pens.Red, x + size, y, x, y + size)

        ' вертикальная линия
        g.DrawLine(Pens.Blue, CInt(x + size / 2), y, CInt(x + size / 2),
        CInt(y + size))
        ' горизонтальная линия
        g.DrawLine(Pens.Blue, CInt(x), CInt(y + size / 2), CInt(x +
        size), CInt(y + size / 2))

        Uzor(x - 3 * size / 4, y + size / 4, size / 2)
        Uzor(x + 5 * size / 4, y + size / 4, size / 2)
        Uzor(x + size / 4, y - 3 * size / 4, size / 2)
        Uzor(x + size / 4, y + 5 * size / 4, size / 2)

        g.Dispose()
    End Sub

    Private Sub mnuUzor_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles mnuUzor.Click
        Refresh()
        Uzor(200, 200, 120)
    End Sub
```

7.3.3. Типы фракталов

Фракталы делятся на группы. Самые большие группы это:

- геометрические фракталы;
- алгебраические фракталы;
- системы итерируемых функций;
- стохастические фракталы.

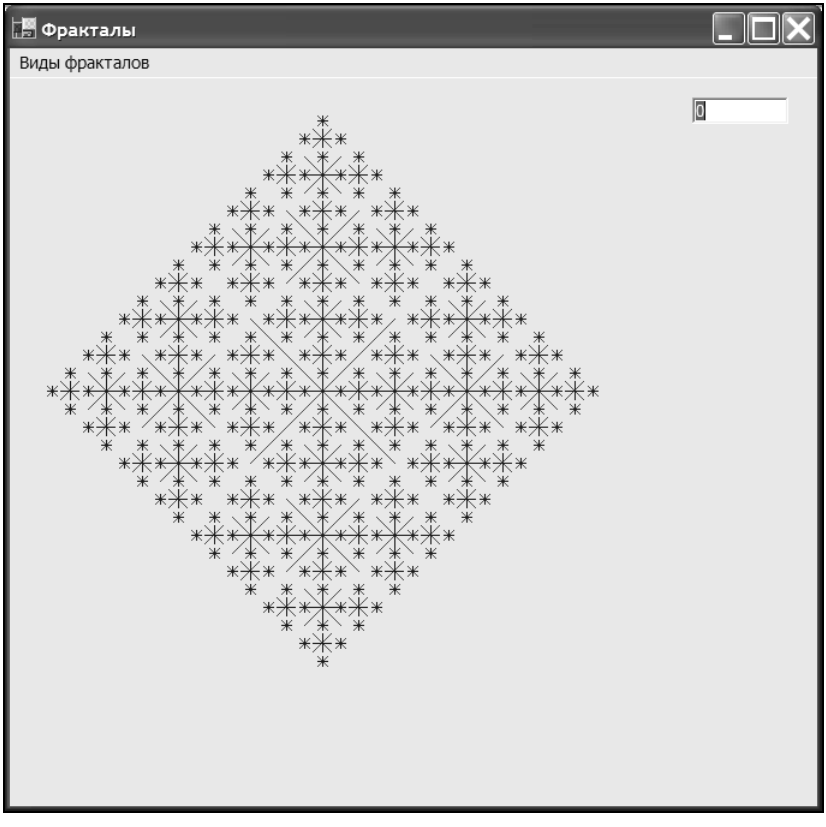

Рис. 7.6. Фрактальная скатерть

7.3.4. Геометрические фракталы

Именно с них и начиналась история фракталов. Этот тип фракталов получается путем простых геометрических построений. Обычно при построении геометрических фракталов поступают так: берется своеобразный шаблон — набор отрезков, на основании которых будет строиться фрактал. Далее к этому шаблону применяют набор правил, который преобразует ее в какую-либо геометрическую фигуру. К каждой получившейся части этой фигуры вновь применяют тот же набор правил. С каждым шагом фигура будет становиться все сложнее и сложнее, и в итоге получим геометрический фрактал. Рассмотренная выше кривая Пеано является геометрическим фракталом. Классические примеры геометрических фракталов — *снежинка Коха*, *Лист*, *Драконова ломаная*, *треугольник Серпинского*.

Снежинка Коха

Из этих геометрических фракталов очень интересным и довольно знаменитым является первый — снежинка Коха. Строится она на основе равностороннего треугольника, каждая сторона которого заменяется на 4 линии каждая длиной в 1/3 исходной. Таким образом, с каждой итерацией длина кривой увеличивается на треть. И если мы сделаем бесконечное число итераций, то получим фрактал — снежинку Коха бесконечной длины (рис. 7.7). Получается, что наша бесконечная кривая покрывает ограниченную площадь. Процесс деления стороны треугольника на три части удобно представить с помощью двух одинаковых треугольников, углы которых ломают стороны нужным нам образом (листинг 7.7).

Листинг 7.7. Создание снежинки Коха

```
'--------------------------------------
' Снежинка Коха © 2004 Александр Климов
'--------------------------------------
    Sub Koch(ByVal xpos, ByVal ypos, ByVal size)
        Dim x()
        Dim y()

        ReDim x(12), y(12)
        Dim shift As Single
        Dim i As Integer
        If size < 20 Then Exit Sub
        shift = size / 8
        x(1) = xpos
        y(1) = ypos + 4 * shift
        x(2) = xpos + shift
        y(2) = ypos + 2 * shift
        x(3) = xpos + 3 * shift
        y(3) = y(2)
        x(4) = xpos + 2 * shift
        y(4) = ypos
        x(5) = x(3)
        y(5) = ypos - 2 * shift
        x(6) = x(2)
```

```
        y(6) = y(5)
        x(7) = xpos
        y(7) = ypos - 4 * shift
        x(8) = xpos - shift
        y(8) = y(5)
        x(9) = xpos - 3 * shift
        y(9) = y(5)
        x(10) = xpos - 2 * shift
        y(10) = ypos
        x(11) = x(9)
        y(11) = y(2)
        x(12) = x(8)
        y(12) = y(2)

        Dim g As Graphics = CreateGraphics()
        'Рисуем первый треугольник
        g.DrawLine(Pens.Red, x(1), y(1), x(5), y(5))
        g.DrawLine(Pens.Red, x(5), y(5), x(9), y(9))
        g.DrawLine(Pens.Red, x(9), y(9), x(1), y(1))

        'Рисуем второй треугольник
        g.DrawLine(Pens.Red, x(3), y(3), x(7), y(7))
        g.DrawLine(Pens.Red, x(7), y(7), x(11), y(11))
        g.DrawLine(Pens.Red, x(11), y(11), x(3), y(3))

        ' Закомментируйте эти строчки,
        ' чтобы увидеть начальные треугольники

        Koch(xpos, ypos, 2 * shift)

        For i = 1 To 12
            Koch(x(i), y(i), 3 * shift)
        Next
        g.Dispose()
    End Sub
```

```
    Private Sub mnuKoch_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles mnuKoch.Click
        Refresh()
        Koch(200, 200, 200)
    End Sub
```

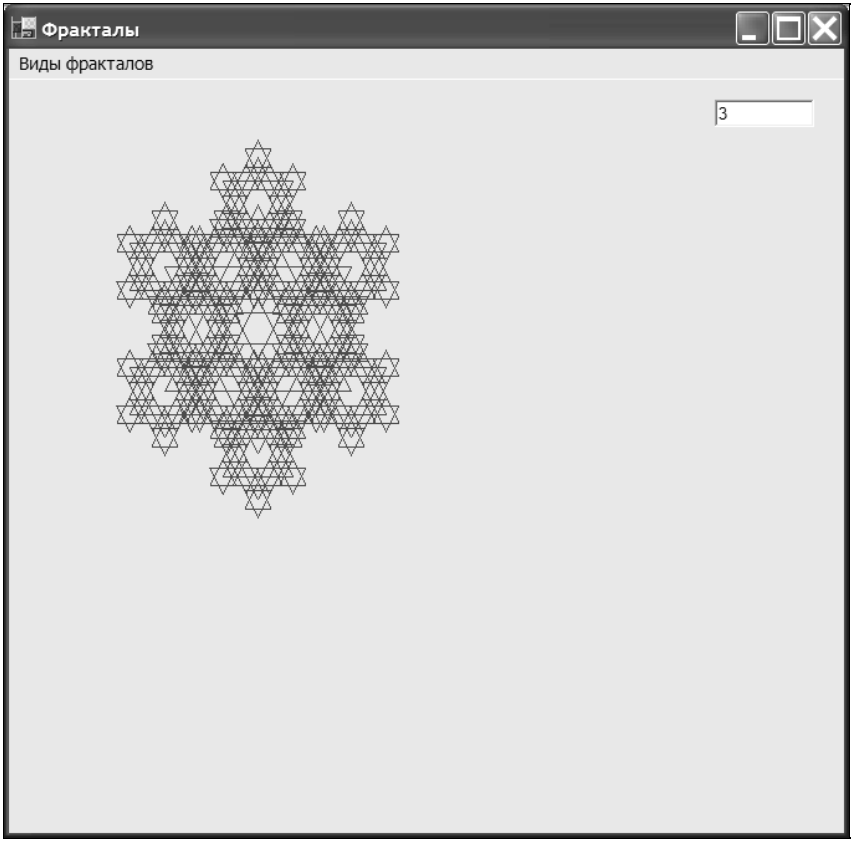

Рис. 7.7. Снежинка Коха

Драконова ломаная

Драконова ломаная относится к классу самоподобных рекурсивно порождаемых геометрических структур. Ломаная нулевого порядка представляет собой просто прямую линию. Согнув ее под прямым углом, мы получим ломаную первого порядка (рис. 7.8). Продолжаем гнуть получаемые отрезки под прямым углом по следующему правилу: первый угол выворачиваем на-

ружу, а второй — вовнутрь. На рис. 7.9 и 7.10 приведены примеры кривых соответственно второго и третьего порядка. Если нарисовать ломаную достаточно большого порядка, то можно получить фигуру, отдаленно напоминающую дракона. Процедура рисования Драконовой ломаной может выглядеть следующим образом (листинг 7.8).

Листинг 7.8. Создание Драконовой ломаной

```
'--------------------------------------
' Драконова ломаная © 2004 Александр Климов
'--------------------------------------
    Sub Dragon(ByRef x1 As Integer, ByRef y1 As Integer, ByRef x2 As
Integer, ByRef y2 As Integer, ByRef k As Integer)
        Dim xn As Integer
        Dim yn As Integer
        Dim g As Graphics = CreateGraphics()

        If k > 0 Then
            xn = (x1 + x2) / 2 + (y2 - y1) / 2
            yn = (y1 + y2) / 2 - (x2 - x1) / 2
            Call Dragon(x1, y1, xn, yn, k - 1)
            Dragon(x2, y2, xn, yn, k - 1)
        Else
            g.DrawLine(Pens.Red, x1, y1, x2, y2)
        End If

        g.Dispose()
    End Sub

    Private Sub mnuDragon_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles mnuDragon.Click
        Refresh()
        Dragon(200, 300, 500, 300, CInt(TextBox1.Text))
    End Sub
```

Чтобы вы могли разглядеть, как выглядит Драконова ломаная при разных коэффициентах порядка, я ввел в код величину, берущую свое значение из текстового поля. Чтобы получить фрактал *Хартера—Хейтуэя* (подмножество Драконовой ломаной), введите в текстовое поле значение 14 (рис. 7.11).

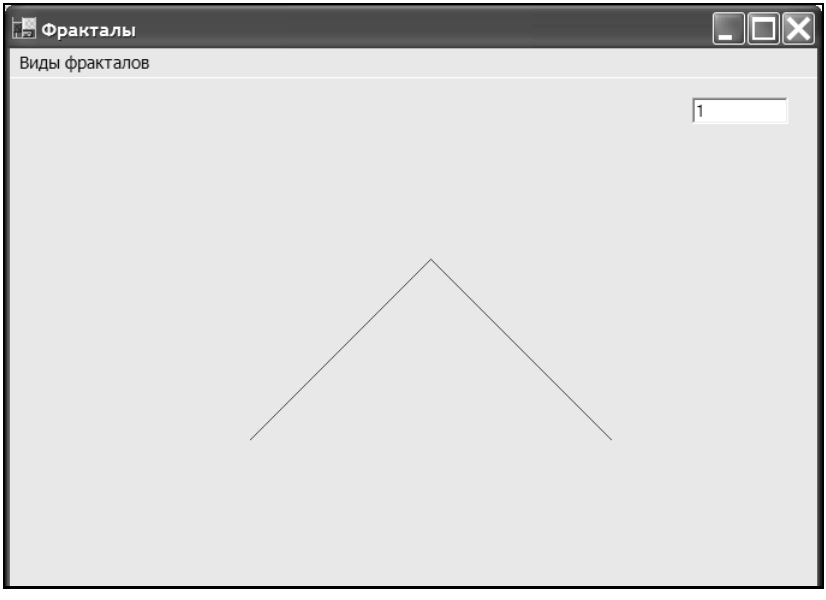

Рис. 7.8. Ломаная первого порядка

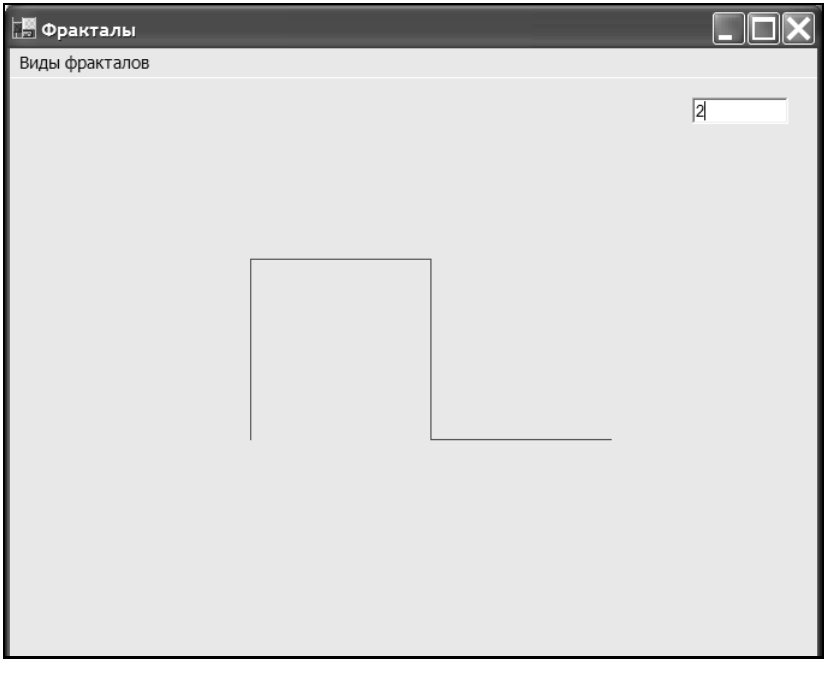

Рис. 7.9. Ломаная второго порядка

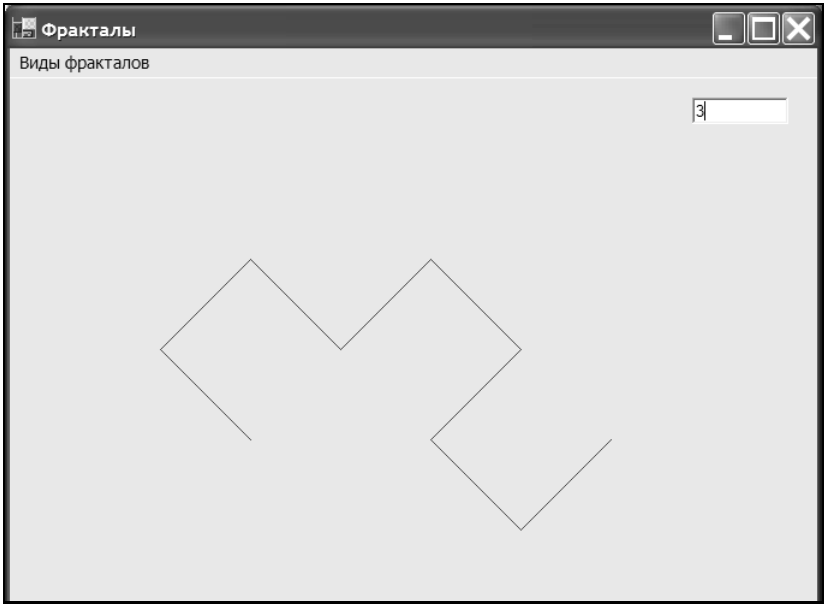

Рис. 7.10. Ломаная третьего порядка

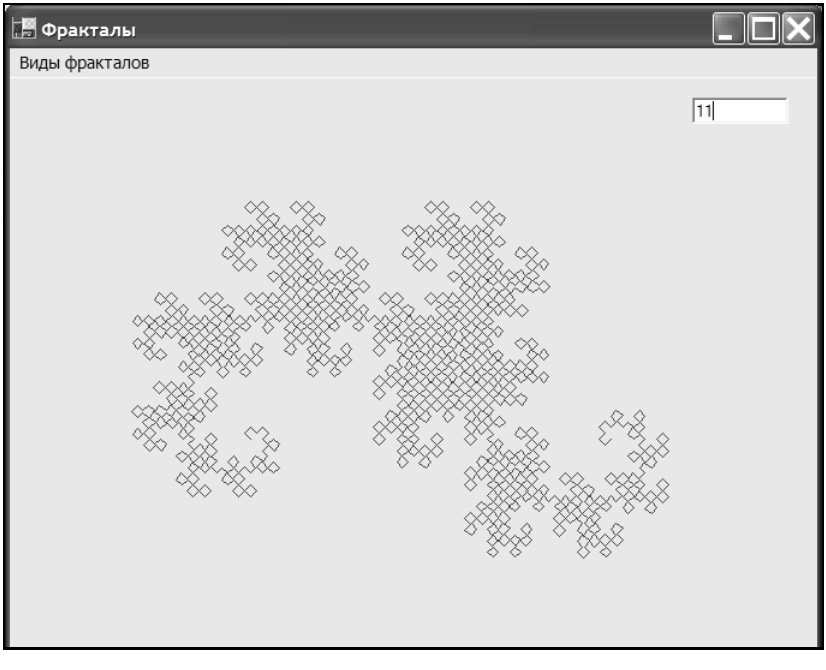

Рис. 7.11. Фрактал Хартера—Хейтуэя

Треугольник Серпинского

В 1915 году польский математик Вацлав Серпинский придумал занимательный объект, известный как треугольник Серпинского. Треугольник Серпинского также называют салфеткой, или решетом Серпинского. Этот треугольник один из самых ранних известных примеров фракталов. Для его построения из центра равностороннего треугольника мысленно вырежем кусок треугольной формы, который своими вершинами будет упираться в середины сторон исходного треугольника. Повторим эту же процедуру для трех образовавшихся треугольников (за исключением центрального) и так до бесконечности. Если мы теперь возьмем любой из образовавшихся треугольников и увеличим его — получим точную копию целого. В данном случае мы имеем дело с полным самоподобием. Попробуем написать программу, рисующую данный треугольник, — проект `Serpinski` (листинг 7.9).

Листинг 7.9. Создание треугольника Серпинского

```
'--------------------------------------
' Треугольник Серпинского © 2004 Александр Климов
'--------------------------------------
    Private Sub DrawTriangle(ByVal x1 As Single, ByVal y1 As Single,
ByVal x2 As Single, ByVal y2 As Single, ByVal x3 As Single, ByVal y3 As
Single)
        ' Рисуем треугольник
        Dim g As Graphics = CreateGraphics()
        g.DrawLine(Pens.Red, x1, y1, x2, y2)
        g.DrawLine(Pens.Red, x2, y2, x3, y3)
        g.DrawLine(Pens.Red, x3, y3, x1, y1)
        g.Dispose()
    End Sub


    Private Sub DrawSerpinski(ByVal x1 As Single, ByVal y1 As Single,
ByVal x2 As Single, ByVal y2 As Single, ByVal x3 As Single, ByVal y3 As
Single, ByVal counter As Integer)
        Dim x1n, y1n, x2n, y2n, x3n, y3n As Single


        If counter > 0 Then
            x1n = (x1 + x2) / 2
            y1n = (y1 + y2) / 2
            x2n = (x2 + x3) / 2
            y2n = (y2 + y3) / 2
            x3n = (x3 + x1) / 2
```

```
            y3n = (y3 + y1) / 2
            DrawTriangle(x1n, y1n, x2n, y2n, x3n, y3n)

            DrawSerpinski(x1, y1, x1n, y1n, x3n, y3n, counter - 1)
            DrawSerpinski(x2, y2, x1n, y1n, x2n, y2n, counter - 1)
            DrawSerpinski(x3, y3, x2n, y2n, x3n, y3n, counter - 1)
        End If
    End Sub

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
        Dim g As Graphics = CreateGraphics()
        g.Clear(BackColor)
        DrawTriangle(320, 10, 600, 470, 40, 470)
        DrawSerpinski(320, 10, 600, 470, 40, 470, TextBox1.Text)
        g.Dispose()
    End Sub
```

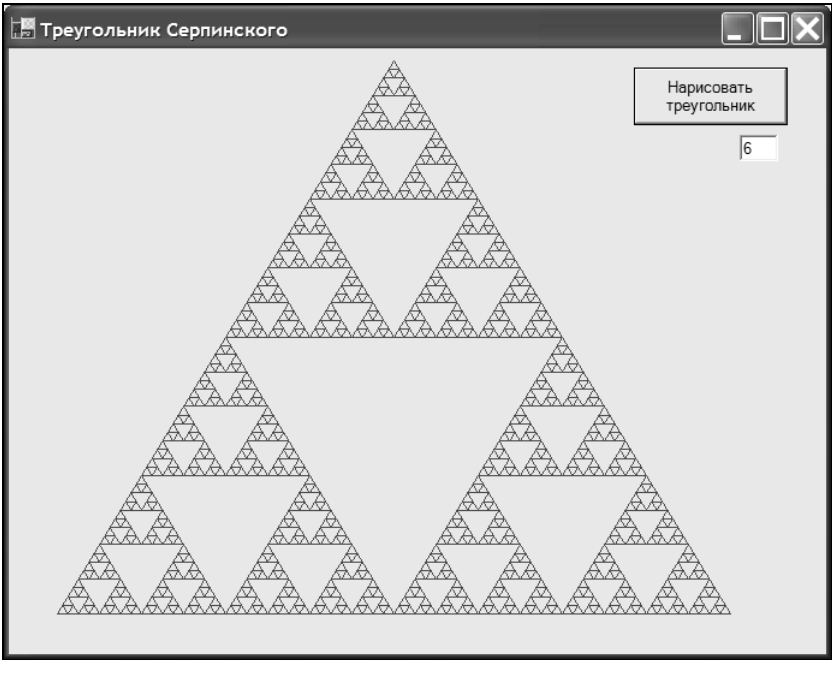

Рис. 7.12. Треугольник Серпинского

Чтобы пример был более наглядным, я решил не задавать жестко число циклов для рисования множества треугольников. Вместо этого я добавил на форму текстовое поле, в котором установил по умолчанию значение 6 (рис. 7.12). Вы можете для детального просмотра построения треугольника последовательно вводить числа 1, 2, 3 и т. д. Имейте в виду, что с увеличением числа возрастает нагрузка на процессор компьютера, занятого вычислениями. Например, у меня компьютер завис, когда я попробовал установить значение, равное 22.

7.3.5. Алгебраические фракталы

Вторая большая группа фракталов — алгебраические. Свое название они получили за то, что их строят на основе алгебраических формул. Чтобы проиллюстрировать алгебраические фракталы, обратимся к классике — множеству Мандельброта. В данном случае я не стал изобретать велосипед и самостоятельно программировать изображение фрактала, так как нашел готовый пример на сайте **http://www.vb-helper.com** (листинг 7.10).

Листинг 7.10. Создание множества Мандельброта

```
'--------------------------------------
' Множество Мандельброта
' Пример взят с сайта http://www.vb-helper.com
' Автор примера Род Стивенс
'--------------------------------------
Imports System.Math

    Private m_DrawingBox As Boolean
    Private m_X1, m_Y1, m_X2, m_Y2 As Integer
    Private m_ZoomingBitmap As Bitmap
    Private m_ZoomingGraphics As Graphics

    ' Графические переменные
    Private m_Bitmap As Bitmap
    Private m_Graphics As Graphics

    ' Координаты
    Private Const MIN_X As Double = -2.2
    Private Const MAX_X As Double = 1
    Private Const MIN_Y As Double = -1.2
    Private Const MAX_Y As Double = 1.2
```

```
    Private m_Wxmin As Double = MIN_X
    Private m_Wxmax As Double = MAX_X
    Private m_Wymin As Double = MIN_Y
    Private m_Wymax As Double = MAX_Y

    Public Const MaxIterations As Integer = 128

    ' Цвета
    Public m_NumColors As Integer
    Private m_Colors() As Color

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        ' Рисуем первое изображение
        ' Загружаем цвета
        m_NumColors = 15
        ReDim m_Colors(m_NumColors)
        For i As Integer = 0 To m_NumColors - 1
            m_Colors(i) = Color.FromArgb( _
                QBColor(i + 1) + &HFF000000)
        Next i

        ' Рисуем фрактал
        DrawMandelbrot()
    End Sub

    Private Sub Form1_Resize(ByVal sender As Object, ByVal e As
System.EventArgs) Handles MyBase.Resize
        ' Перерисовываем при изменении размера формы
        If m_Colors Is Nothing Then
            m_NumColors = 15
            ReDim m_Colors(m_NumColors)
            For i As Integer = 0 To m_NumColors - 1
                m_Colors(i) = Color.FromArgb( _
                    QBColor(i + 1) + &HFF000000)
            Next i
        End If

        If Me.WindowState = FormWindowState.Minimized Then Exit Sub
```

```
        ' Если просто перемещение формы, но не изменение размеров
        If Not (m_Bitmap Is Nothing) Then
            If m_Bitmap.Width = PictureBox1.ClientSize.Width And _
               m_Bitmap.Height = PictureBox1.ClientSize.Height _
                Then Exit Sub
        End If

        ' Рисуем фрактал
        DrawMandelbrot()
    End Sub

    Private Sub PictureBox1_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles PictureBox1.Click

    End Sub

    Private Sub PictureBox1_MouseDown(ByVal sender As Object, ByVal e As
System.Windows.Forms.MouseEventArgs) Handles PictureBox1.MouseDown
        ' Начало рисования резинового прямоугольника
        m_DrawingBox = True
        m_X1 = e.X
        m_Y1 = e.Y
        m_X2 = m_X1
        m_Y2 = m_Y1

        ' Делаем копию текущего изображения
        m_ZoomingBitmap = New Bitmap(m_Bitmap)
        m_ZoomingGraphics = Graphics.FromImage(m_ZoomingBitmap)
    End Sub

    Private Sub PictureBox1_MouseMove(ByVal sender As Object, ByVal e As
System.Windows.Forms.MouseEventArgs) Handles PictureBox1.MouseMove
        ' Продолжаем рисовать резиновый прямоугольник
        If Not m_DrawingBox Then Exit Sub

        ' Сохраняем углы
        m_X2 = e.X
        m_Y2 = e.Y
```

```
        ' Рисуем новый прямоугольник
        Dim gr As Graphics = PictureBox1.CreateGraphics()
        Dim rect As New Rectangle
        rect.X = Min(m_X1, m_X2)
        rect.Y = Min(m_Y1, m_Y2)
        rect.Width = Abs(m_X2 - m_X1)
        rect.Height = Abs(m_Y2 - m_Y1)
        m_ZoomingGraphics.DrawImage(m_Bitmap, 0, 0)
        m_ZoomingGraphics.DrawRectangle(Pens.White, rect)

        ' Выводим результат
        PictureBox1.Image = m_ZoomingBitmap
    End Sub

    Private Sub PictureBox1_MouseUp(ByVal sender As Object, ByVal e As
System.Windows.Forms.MouseEventArgs) Handles PictureBox1.MouseUp
        ' Заканчиваем рисование резинового прямоугольника
        m_DrawingBox = False

        ' Освобождаем ресурсы
        PictureBox1.Image = Nothing
        m_ZoomingBitmap.Dispose()
        m_ZoomingGraphics.Dispose()
        m_ZoomingBitmap = Nothing
        m_ZoomingGraphics = Nothing

        ' Сохраняем новые углы
        m_X2 = e.X
        m_Y2 = e.Y

        ' Кладем выбранные координаты по порядку
        Dim x1, x2, y1, y2, factor As Double
        x1 = Min(m_X1, m_X2)
        x2 = Max(m_X1, m_X2)
        y1 = Min(m_Y1, m_Y2)
        y2 = Max(m_Y1, m_Y2)

        ' Конвертируем экранные координаты
        factor = (m_Wxmax - m_Wxmin) / PictureBox1.ClientSize.Width
```

```
    m_Wxmax = m_Wxmin + x2 * factor
    m_Wxmin = m_Wxmin + x1 * factor

    factor = (m_Wymax - m_Wymin) / PictureBox1.ClientSize.Height
    m_Wymax = m_Wymin + y2 * factor
    m_Wymin = m_Wymin + y1 * factor

    ' Рисуем фрактал
    DrawMandelbrot()
End Sub

' Подгоняем размеры
Private Sub AdjustAspect()
    Dim want_aspect, Picturebox1_aspect As Double
    Dim hgt, wid, mid As Double

    want_aspect = (m_Wymax - m_Wymin) / (m_Wxmax - m_Wxmin)
    Picturebox1_aspect = _
      PictureBox1.ClientSize.Height / PictureBox1.ClientSize.Width
    If want_aspect > Picturebox1_aspect Then
        wid = (m_Wymax - m_Wymin) / Picturebox1_aspect
        mid = (m_Wxmin + m_Wxmax) / 2
        m_Wxmin = mid - wid / 2
        m_Wxmax = mid + wid / 2
    Else
        hgt = (m_Wxmax - m_Wxmin) * Picturebox1_aspect
        mid = (m_Wymin + m_Wymax) / 2
        m_Wymin = mid - hgt / 2
        m_Wymax = mid + hgt / 2
    End If
End Sub

' Процедура рисования фрактала
Private Sub DrawMandelbrot()
    ' Ничего не делаем, если графика имеет нулевой размер
    If PictureBox1.ClientSize.Width < 1 Or _
       PictureBox1.ClientSize.Height < 1 Then Exit Sub
```

```
        ' Создаем графические объекты
        If Not (m_Graphics Is Nothing) Then
            PictureBox1.Image = Nothing
            m_Graphics.Dispose()
            m_Bitmap.Dispose()
        End If
        m_Bitmap = New Bitmap(PictureBox1.ClientSize.Width,
PictureBox1.ClientSize.Height)
        m_Graphics = Graphics.FromImage(m_Bitmap)

        ' Очищаем картинку
        m_Graphics.Clear(Color.Black)
        PictureBox1.Image = m_Bitmap
        PictureBox1.Refresh()

        Const MAX_MAG_SQUARED As Double = 4.0

        AdjustAspect()

        Dim clr As Integer
        Dim ReaC, ImaC, dReaC, dImaC, ReaZ, ImaZ, ReaZ2, ImaZ2 As Double

        Dim wid As Integer = PictureBox1.ClientSize.Width
        Dim hgt As Integer = PictureBox1.ClientSize.Height
        dReaC = (m_Wxmax - m_Wxmin) / (wid - 1)
        dImaC = (m_Wymax - m_Wymin) / (hgt - 1)

        ReaC = m_Wxmin
        For i As Integer = 0 To wid - 1
            ImaC = m_Wymin
            For j As Integer = 0 To hgt - 1
                ReaZ = 0
                ImaZ = 0
                ReaZ2 = 0
                ImaZ2 = 0
                clr = 0
                Do While clr < MaxIterations And _
                        ReaZ2 + ImaZ2 < MAX_MAG_SQUARED
                    ReaZ2 = ReaZ * ReaZ
```

```
                ImaZ2 = ImaZ * ImaZ
                ImaZ = 2 * ImaZ * ReaZ + ImaC
                ReaZ = ReaZ2 - ImaZ2 + ReaC
                clr = clr + 1
            Loop

            m_Bitmap.SetPixel(i, j, _
                m_Colors(clr Mod m_NumColors))

            ImaC = ImaC + dImaC
        Next j
        ReaC = ReaC + dReaC

        If i Mod 10 = 0 Then PictureBox1.Refresh()
    Next i

    PictureBox1.Refresh()

    Me.Text = "Mandelbrot (" & _
        Format$(m_Wxmin) & ", " & _
        Format$(m_Wymin) & ")-(" & _
        Format$(m_Wxmax) & ", " & _
        Format$(m_Wymax) & ")"
End Sub


    Private Sub mnuFullScale_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles mnuFullScale.Click
        m_Wxmin = MIN_X
        m_Wxmax = MAX_X
        m_Wymin = MIN_Y
        m_Wymax = MAX_Y

        DrawMandelbrot()
    End Sub
```

На рис. 7.13, изображающем множество Мандельброта, можно взять небольшой участок и увеличить его до размеров всего экрана (как в микроскоп). Что же мы увидим? Проявление самоподобности. Не точной самопо-

добности, но близкой, и с ней мы будем сталкиваться постоянно, увеличивая части нашего фрактала больше и больше.

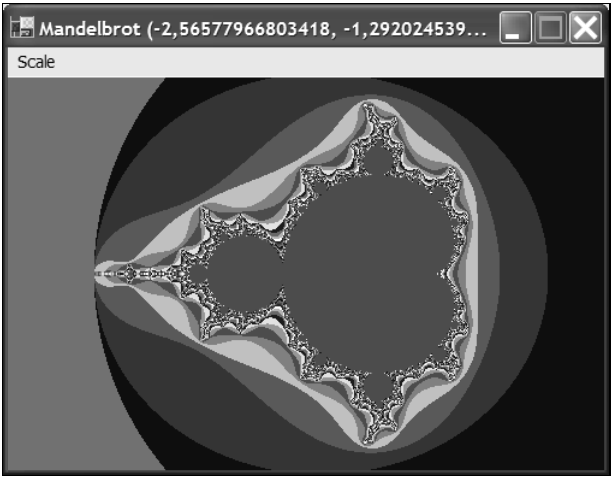

Рис. 7.13. Множество Мандельброта

7.3.6. L-системы

Любителям фракталов и математических картинок известны фантастические изображения растений, полученные с помощью специальных программ. Это так называемые *L-системы*. В основе их построения лежат два принципа. Первый — это так называемая "черепашья графика" (оператор `draw`) патриарха GWBASIC и его детей Turbo Basic и QBasic, когда движение рисуется пошагово в приращениях относительно текущей точки, либо данное поведение моделируется заданием движения в приращениях координат. Второй принцип — изюминка метода: каждое единичное движение заменяется на весь рисунок. Например, нарисуем вилку-рогатульку. На следующем шаге работы программы каждая из трех палочек вилки заменяется такой же вилкой, превращая вилку в ветку с сучками. После следующего шага получим лохматый куст, потом пушистое дерево, красивое, фрактальное. Меняя вид начальной картинки, можно получать самые разные изображения от зонтиков укропа до колючего перекати-поля или пучка водорослей.

Суть L-кодирования сводится к следующему. Представим себе некое виртуальное программируемое устройство, состоящее из пера, управляющего им механизма и листа бумаги. Управляющий пером механизм способен исполнять несколько команд. А именно: он может опустить перо на бумагу и вычертить прямой отрезок заданной длины в направлении текущей ориентации пера (команда `F`). Он может изменить ориентацию пера по отношению

к текущей на какой-то заданный относительный угол по часовой или против часовой стрелки (команды + и -). Он может также запоминать (заносить в стек) свое текущее состояние (команда [) и вспоминать (извлекать из стека) ранее запомненное состояние (команда]). Под состоянием в данном случае понимается тройка чисел (x, y, a), где x и y — это координаты пера и a — угол, определяющий направление ориентации пера. Таким образом, задав некое начальное направление, определив относительный угол поворота в 90 градусов и задав длину отрезка, при помощи последовательности команд F+F+F+F мы можем нарисовать квадрат. Определив относительный угол поворота в 60 градусов, при помощи последовательности команд F++F++F можно нарисовать равносторонний треугольник.

Предположим также, что в программы для нашего виртуального устройства, кроме пяти перечисленных команд, можно включать любые другие символы, которые управляющий механизм будет просто игнорировать. То есть, если мы введем программу F+BF+CCF+CF, то устройство все равно нарисует квадрат. Теперь мысленно оснастим наше устройство приставкой, которая перед тем как передать введенную программу на управляющий механизм, может заданное число раз просматривать ее и при каждом очередном просмотре заменять любые символы последовательности по предварительно указанным правилам. Исходную программную последовательность символов теперь будем называть *аксиомой*. Например, введем аксиому FB+ и определим правило B < F+FB. Зададим также количество просмотров, равное, например, двум. Тогда на входе механизма после обработки введенной аксиомы приставкой получим последовательность FF+FF+FB+. Вот, собственно, и все. При помощи описанного несложного виртуального устройства можно строить множество самых разнообразных фрактальных форм — от традиционных математических фракталов, таких как, например, снежинка Коха или кривая Гильберта, до структур, очень напоминающих растительную или подводную жизнь. Рассмотрим, как кодируются L-системы в общепринятых обозначениях.

Движение вперед обозначается буквой F (Forward — вперед), поворот по часовой стрелке обозначим "+", а против — "-", причем само значение поворота задается в программе и постоянно для всех движений. Буквой B (Back — назад) обозначается возврат без прорисовки, нам это не пригодится.

Для нас важнее механизм возврата. Точку, в которую надо возвращаться, обозначим "[", а место, откуда происходит возврат — "]". Тогда вилка будет иметь вид:

- F — движение вперед;
- [— запомнить позицию;
- + — поворот вправо на 22,5 градусов;
- F — движение вперед после поворота;

-] — возврат в запомненную позицию;
- [— запомнить позицию;
- \- — поворот влево относительно направления в запомненной точке;
- F — движение вперед после поворота;
-] — возврат в запомненную позицию.

Такое движение уже поддается кодированию. Можно реализовать и более сложную операцию — закодировать и следующий шаг — замену каждого прямого отрезка на такую же вилку. Два шага нарисуют три шага, три шага — четыре шага. Прорисовывать каждый шаг заменой текста программы довольно утомительно, и мы вспоминаем о рекурсии. Самое подходящее для нее место! Выполнив необходимые объявления переменных и передачу значений координат точек возврата, мы добиваемся, что любое дерево рисуется по заранее заданной формуле одной маленькой процедуркой, которая сама себя и вызывает. Представленная на ваш суд программа — проект LSystem (листинг 7.11) — позволит вам с восхищением (ибо порядок прорисовки фрактально-непредсказуем) отслеживать на экране рост дерева. Запустив программу, вы увидите как она нарисует ветку, наклоняемую ветром. Меняя переменную Kmax, можно уменьшать или увеличивать глубину рекурсии, или, что тоже самое, "пушистость" ветки (рис. 7.14). А меняя закон движения, можно получать самые удивительные и фантастические изображения.

Листинг 7.11. Создание L-системы

```
'--------------------------------------
' L-система © 2004 Александр Климов
'--------------------------------------
    Dim F(50) As String
    Dim X(10) As Single
    Dim Y(10) As Single
    Dim Fi(10) As Single
    Dim Xtemp(10) As Single
    Dim Ytemp(10) As Single
    Dim Fitemp(10) As Single
    Dim Xtemp2(10) As Single
    Dim Ytemp2(10) As Single
    Dim Fitemp2(10) As Single
    Dim Fi0 As Single
    Dim k As Integer
    Dim Kmax As Integer
    Dim vetka As Integer
```

```
    Dim i(10) As Integer
    Dim Xnew As Single
    Dim Ynew As Single

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        k = 1
        Kmax = 5  ' количество циклов

        vetka = 300 / Kmax ^ 2.5  ' размер элемента
        X(0) = 800  ' начальная координата точки рисования
        Y(0) = 600
        Fi(0) = 0 ' начальный угол рисования
        Fi0 = 3.14 / 8 ' угол поворота при рисовании

        ' программа для прорисовки
        F(1) = "-"   ' так обозначаем поворот налево
        F(2) = "f"   ' так обозначаем шаг вперед
        F(3) = "+"   ' так обозначаем поворот направо
        F(4) = "f"
        F(5) = "+"
        F(6) = "["   ' запоминаем текущее положение для возврата к нему
                       по знаку "]"
        F(7) = "+"
        F(8) = "f"
        F(9) = "-"
        F(10) = "f"
        F(11) = "-"
        F(12) = "]"  ' обозначаем возврат к положению, обозначенному
                       знаком "["
        F(13) = "-"
        F(14) = "["
        F(15) = "-"
        F(16) = "f"
        F(17) = "+"
        F(18) = "f"
        F(19) = "+"
        F(20) = "f"
        F(21) = "]"
```

```
    End Sub
    Sub Cicl()

        X(k) = X(k - 1)
        Y(k) = Y(k - 1)
        Fi(k) = Fi(k - 1)

        For i(k) = 1 To 50
            If F(i(k)) = "-" Then Fi(k) = Fi(k) - Fi0
            If F(i(k)) = "f" Then
                If k = Kmax Then
                    Drw()
                    X(k) = Xnew
                    Y(k) = Ynew
                Else
                    k = k + 1
                    Cicl()
                    k = k - 1
                    X(k) = X(k + 1)
                    Y(k) = Y(k + 1)
                End If
            End If

            If F(i(k)) = "+" Then Fi(k) = Fi(k) + Fi0
            If F(i(k)) = "[" Then Xtemp(k) = X(k) : Ytemp(k) = Y(k) :
Fitemp(k) = Fi(k)
            If F(i(k)) = "]" Then X(k) = Xtemp(k) : Y(k) = Ytemp(k) :
Fi(k) = Fitemp(k)
            If F(i(k)) = "[[" Then Xtemp2(k) = X(k) : Ytemp2(k) = Y(k) :
Fitemp2(k) = Fi(k)
            If F(i(k)) = "]]" Then X(k) = Xtemp2(k) : Y(k) = Ytemp2(k) :
Fi(k) = Fitemp2(k)

        Next i(k)
    End Sub

    Sub Drw()
        Xnew = X(k) + vetka * Math.Sin(Fi(k))
        Ynew = Y(k) - vetka * Math.Cos(Fi(k))
        Dim g As Graphics = CreateGraphics()
        g.TranslateTransform(-450, -150)
```

```
        g.DrawLine(Pens.Green, X(k), Y(k), Xnew, Ynew)
        g.Dispose()

    End Sub

    Private Sub Timer1_Tick(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Timer1.Tick
        Cicl()
        Timer1.Enabled = False
    End Sub
```

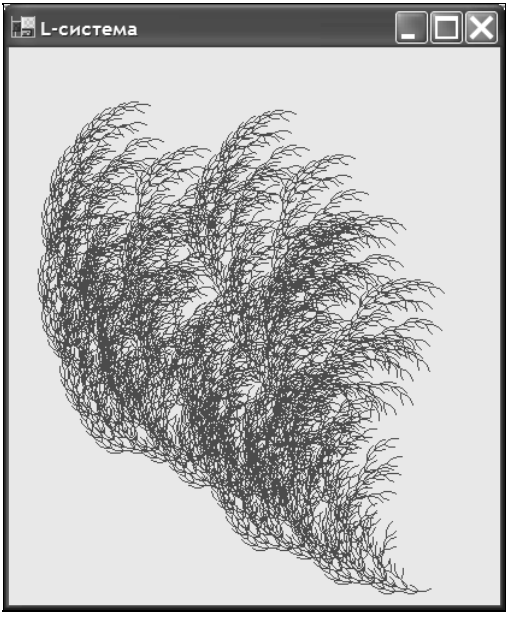

Рис. 7.14. Веточка, нарисованная с помощью L-системы

Надо заметить, что данная программа является практически полным аналогом проекта, написанного на VB 6.0. Мне пришлось только поменять одну строчку для вывода линии с помощью метода `DrawLine`, что говорит о глубокой родственной связи VB. NET 2003 со старой версией бейсика.

7.4. Советы и рекомендации

Приглядитесь к окружающим вами предметам и попытайтесь программно нарисовать их в вашей программе. Особенно интересными могут оказаться фигуры, получаемые с помощью L-систем.

Глава 8

Изображения

GDI+ имеет мощную поддержку изображений с помощью класса `Bitmap`. У этого класса очень много свойств и методов. Я выбрал из них те, которые были практически недоступны в VB 6.0.

8.1. Сглаживание рисунков

Одной из важных и полезных возможностей класса `Graphics` является способность сглаживать рисунки при изменении их размеров. Вероятно, вам не раз приходилось увеличивать или уменьшать размеры изображений в графических редакторах. При этом вы могли видеть искажения, получаемые после таких операций. В "продвинутых" редакторах существует возможность сглаживания таких искажений путем обработки резких переходов цвета. Рассмотрим такой случай на практическом примере. Загрузим в проект тестовую картинку, изменим ее размеры и выведем полученный результат на экран. Для демонстрации примера я создал проект `Antialias` и добавил на главную форму проекта три графических поля и две кнопки. В одно из трех графических полей я загрузил тестовую картинку. Два остальных графических поля остались пустыми. Попробуем сначала уменьшить в два раза тестовую картинку и скопировать ее в элемент `picNoChange` без каких-либо изменений (листинг 8.1).

Листинг 8.1. Пример искажения рисунка

```
'--------------------------------------
' Сглаживание © 2004 Александр Климов
'--------------------------------------

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click

        ' Картинка-источник
        Dim img As New Bitmap(picSource.Image)
```

```
        ' Размеры для новой картинки
        Dim wid As Integer = picSource.Width \ 2
        Dim hgt As Integer = picSource.Height \ 2
        Dim bmp_to As New Bitmap(wid, hgt)

        ' Копируем картинку с уменьшенными размерами без изменений
        Dim g As Graphics = Graphics.FromImage(bmp_to)
        g.DrawImage(img, 0, 0, wid - 1, hgt - 1)

        ' Выводим результат
        picNoChange.Image = bmp_to
    End Sub
```

Как видите, при изменении своих размеров, картинка потеряла свою четкость (рис. 8.1). Попробуем теперь применить один из методов сглаживания (листинг 8.2).

Листинг 8.2. Пример сглаживания рисунка

```
    Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button2.Click
        Dim img As New Bitmap(picSource.Image)

        Dim wid As Integer = picSource.Width \ 2
        Dim hgt As Integer = picSource.Height \ 2
        Dim bmp_to As New Bitmap(wid, hgt)

        ' Копируем картинку с режимом сглаживания
        Dim g As Graphics = Graphics.FromImage(bmp_to)
        g.InterpolationMode = _
           Drawing2D.InterpolationMode.HighQualityBilinear
        g.DrawImage(img, 0, 0, wid - 1, hgt - 1)
        picChange.Image = bmp_to
    End Sub
```

Рис. 8.1. Сглаживание рисунков

8.2. Метод *MakeTransparent*

К числу интересных методов, доступных для класса `Bitmap`, относится метод `MakeTransparent`. Этот метод позволяет назначить указанный цвет прозрачным для картинки. Вот простейшая реализация примера. В стандартном графическом редакторе Paint из пакета Windows я залил зеленым цветом всю будущую картинку и вывел на ней название издательства, согласившегося выпустить эту несерьезную книжку, которую вы держите в руках. Теперь в новом проекте `MakeTransparent` я разместил два элемента `PictureBox` с именами `picFrom` и `picTo`. Свойству `Image` элемента `picFrom` я назначил свою нарисованную картинку, которая и будет объектом нашего исследования. Также на форму я поместил две кнопки, чьи свойства оставил по умолчанию. Условно, зеленый цвет картинки можно назвать его фоном (так как он занимает бóльшую часть картинки). Попробуем сделать цвет фона прозрачным (рис. 8.2). Для этого служит следующий код (листинг 8.3).

Листинг 8.3. Использование прозрачного цвета

```
'---------------------------------------
' Прозрачный цвет картинки © 2004 Александр Климов
'---------------------------------------
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
        Dim bmp_fr As New Bitmap(picFrom.Image)
        Dim g As Graphics = Graphics.FromImage(bmp_fr)
        Dim backColor As Color = bmp_fr.GetPixel(3, 3)
```

```
        ' Сделаем выбранный цвет фона прозрачным.
        bmp_fr.MakeTransparent(backColor)

        ' Выводим результат на экран
        picTo.Image = bmp_fr
    End Sub
```

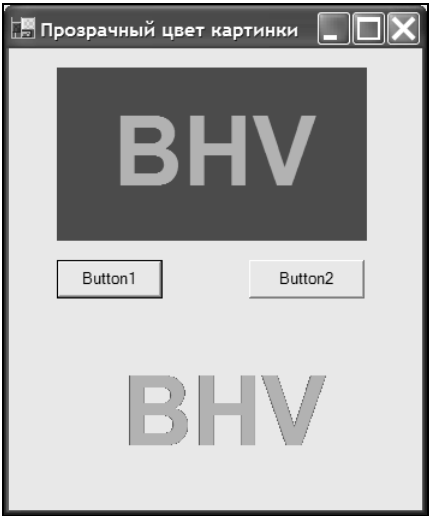

Рис. 8.2. Использование прозрачности для фона картинки

Чтобы получить цвет фона, я использовал метод `GetPixel`, который позволяет узнать цвет в заданной точке. Так как бóльшая часть картинки залита зеленым цветом, то с уверенностью можно сказать, что в точке с координатами (3, 3) будет содержаться нужный нам зеленый цвет. После нажатия на кнопку **Button1** графический объект назначает зеленый цвет прозрачным, и на экране мы видим только текст без фона. Возникает вопрос, а можно ли поступить наоборот и сделать прозрачным цвет букв? Конечно можно (рис. 8.3). Для этого я предусмотрительно поместил на форму вторую кнопку (листинг 8.4).

Листинг 8.4. Другой способ применения прозрачного цвета

```
    Private Sub Button2_Click(ByVal sender As Object, ByVal e As
System.EventArgs) Handles Button2.Click
        Dim bmp_fr As New Bitmap(picFrom.Image)
        Dim g As Graphics = Graphics.FromImage(bmp_fr)
```

```
        ' В этих координатах находится одна из точек букв
        Dim foreColor As Color = bmp_fr.GetPixel(60, 47)

        ' Делаем эту точку прозрачной
        bmp_fr.MakeTransparent(foreColor)

        ' Выводим результат на экран
        picTo.Image = bmp_fr
    End Sub
```

Координату точки буквы я нашел с помощью все той же программы Paint. Открыв графический редактор и поместив указатель мыши на текст, я увидел в нижней строке редактора координаты мыши. Запомнив эти координаты, я использовал их далее в своем примере.

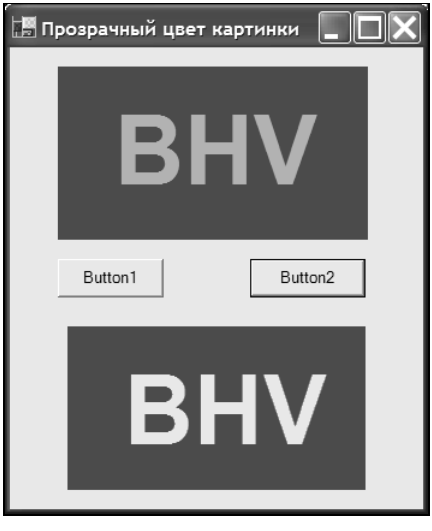

Рис. 8.3. Использование прозрачности для букв

8.2.1. Наложение картинки

Если вы поняли, как сделать прозрачным любой цвет в картинке, то без труда поймете следующий пример. Идея примера состоит в следующем. Допустим, у нас имеются две картинки — одна с изображением чего-либо, вторая — содержит "голый" текст, как в примере, который мы только что

рассматривали. Если просто наложить текст на изображение, то результат будет выглядеть удручающе. А что если убрать фон картинки с текстом (сделать прозрачным) и наложить только текст? Давайте попробуем. Реализация примера содержится в проекте `MakeTransparent2` (листинг 8.5).

Листинг 8.5. Наложение текста на картинку

```
'--------------------------------------
' Прозрачность-2 © 2004 Александр Климов
'--------------------------------------
    Private Sub butMakeTrans_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles butMakeTrans.Click
        ' Работаем с картинкой, содержащей текст
        Dim bmp_txt As New Bitmap(picText.Image)
        Dim g As Graphics = Graphics.FromImage(bmp_txt)
        Dim backColor As Color = bmp_txt.GetPixel(1, 1)

        ' Сделаем выбранный цвет фона прозрачным.
        bmp_txt.MakeTransparent(backColor)

        ' Копируем текст в основную картинку.
        Dim bmp_cat As New Bitmap(picImage.Image)
        Dim gr As Graphics = Graphics.FromImage(bmp_cat)
        gr.DrawImage(bmp_txt, _
            (bmp_cat.Width - bmp_txt.Width) \ 2, _
            bmp_cat.Height - bmp_txt.Height)

        ' Выводим результат на экран.
        picImage.Image = bmp_cat

    End Sub
```

Сначала мы делаем прозрачным цвет фона у картинки, содержащей текст (см. предыдущий пример). Затем копируем получившуюся картинку в основную картинку, содержащую изображение кота. У нас получилось что-то вроде эффекта фотомонтажа (рис. 8.4).

Рис. 8.4. Наложение текста на картинку

8.2.2. Водяные знаки

Можно немного доработать пример и накладывать полупрозрачную картинку. В этом случае создается эффект водяного знака. Водяные знаки нам известны, прежде всего, как одна из степеней защиты бумажных купюр от подделки. Когда деревья были большими, я очень любил рассматривать советские бумажные рубли на свет. На мелких купюрах можно было разглядеть звездочки, а на крупных — профиль вождя мирового пролетариата В. И. Ленина. Водяные знаки применяются широко и в графических файлах как элемент защиты интеллектуальной собственности. Пример с водяным знаком (проект `Watermark`) очень похож на предыдущий пример наложения картинки; кроме того, подробные комментарии в коде помогут вам разобраться с этим примером (листинг 8.6).

Листинг 8.6. Создание водяных знаков

```
'--------------------------------------
' Водяные знаки © 2004 Александр Климов
'--------------------------------------
    ' Процедура нанесения водяного знака на картинку.
    Private Sub DrawWatermark(ByVal watermark As Bitmap, _
        ByVal basic As Bitmap, ByVal x As Integer, ByVal y _
        As Integer)
```

```
        ' Коэффициент прозрачности альфа-канала
        Const ALPHA As Byte = 85

        ' Проходим в цикле через все пикселы картинки,
        ' которая служит водяным знаком,
        ' и заменяем альфа-составляющую цвета,
        ' делая их на треть прозрачнее
        Dim clr As Color
        For py As Integer = 0 To watermark.Height - 1
            For px As Integer = 0 To watermark.Width - 1
                clr = watermark.GetPixel(px, py)
                watermark.SetPixel(px, py, _
                    Color.FromArgb(ALPHA, clr.R, clr.G, clr.B))
            Next px
        Next py

        ' Сделаем цвет фона картинки водяного знака прозрачным
        ' Для этого берем пиксел из верхнего левого угла
        Dim backColor As Color = watermark.GetPixel(3, 3)
        ' И устанавливаем данный цвет прозрачным
        watermark.MakeTransparent(backColor)

        ' Копируем полученный результат в базовую картинку
        Dim g As Graphics = Graphics.FromImage(basic)
        g.DrawImage(watermark, x, y)
    End Sub

    Private Sub butWatermark_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles butWatermark.Click
        ' Картинка в качестве водяного знака
        Dim watermarkImage As New Bitmap(picWatermark.Image)

        ' Основная картинка для опытов
        Dim basicImage As New Bitmap(picImage.Image)

        ' Координаты для вывода водяного знака
        Dim x As Integer = basicImage.Width - watermarkImage.Width
        Dim y As Integer = basicImage.Height - watermarkImage.Height
```

```
        watermarkImage = New Bitmap(picWatermark.Image)

        ' Помещаем водяной знак на базовую картинку
        DrawWatermark(watermarkImage, basicImage, x, y)

        ' Выводим результат на экран
        picImage.Image = basicImage

    End Sub
```

Запустив проект и нажав на кнопку, вы получите изображение водяного знака на заданной картинке (рис. 8.5). Я выбрал одну треть прозрачности для водяного знака. Естественно, вы можете установить собственный коэффициент прозрачности.

Рис. 8.5. Создание водяных знаков

8.3. Применение эффектов

Просто разглядывать картинки не совсем интересно. Гораздо интереснее применить к ним специальные эффекты. Такие графические редакторы, как Adobe Photoshop или Paint Shop Pro, имеют множество эффектов, применимых к картинкам. В основе этих эффектов лежит обработка отдельных участков изображения по особо заданному алгоритму. Рассмотрим несколько

таких алгоритмов. Картинку можно рассматривать как совокупность множества пикселов разного цвета. Узоры из разноцветных пикселов складываются в красивую картинку на экране. С помощью метода `GetPixel` можно получить доступ к любой точке картины и определить ее характеристики. А, имея интересующие нас данные, с помощью метода `SetPixel` уже можно модифицировать картинку. Имейте в виду, что при работе с картинками большого объема обработка всех пикселов может растянуться во времени (особенно на слабых компьютерах). В проекте `Pixel` показано, как можно из цветной картинки получить ее серый вариант (листинг 8.7).

Листинг 8.7. Преобразование изображения в серый цвет

```
'--------------------------------------
' Пикселы © 2004 Александр Климов
'--------------------------------------
    Private Sub butGrayScale_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles butGrayScale.Click
        Dim clr As Integer ' цвет
        Dim wid As Integer ' ширина
        Dim hei As Integer ' высота

        ' Координаты пикселов
        Dim x As Integer
        Dim y As Integer

        ' Создадим объект Bitmap из графического поля.
        Dim myBitmap As Bitmap = picImage.Image

        wid = myBitmap.Width - 1
        hei = myBitmap.Height - 1

        ' Проходим в цикле через все пикселы картинки
        ' и меняем их цвета на серый.
        For y = 0 To hei
            For x = 0 To wid
                With myBitmap.GetPixel(x, y)
                    clr = 0.3 * .R + 0.6 * .G + 0.1 * .B
                End With
                myBitmap.SetPixel(x, y, _
                    Color.FromArgb(255, clr, clr, clr))
```

```
            Next x
        Next y

        ' Выводим результат
        picImage.Image = myBitmap

    End Sub
```

Преобразование изображения в серый цвет осуществляется по специальной формуле

```
0.299RED+0.587GREEN+0.114Blue
```

Данная формула основана на особенностях восприятия человеческого глаза, она положена в основу телевизионного стандарта NTSC (North America Television Standards Committee) и широко используется в графике.

8.3.1. Цветовая модель

Существует и другой способ преобразования изображений — с помощью метода `SetColorMatrix` класса `ColorMatrix`. Этот класс, определенный в пространстве имен `System.Drawing.Imaging`, позволяет видоизменять цветовую модель при помощи матрицы 5×5 преобразования цвета в формате RGBA. Каждый элемент этой матрицы является коэффициентом в диапазоне от 0 до 1. Полная интенсивность приравнивается к 1 (т. е. величина 255 соотносится с 1). Возникает вопрос — почему у нас матрица 5×5, а не 4×4 в соответствии со стандартным представлением цвета? Дело в том, что последняя величина используется при нелинейных преобразованиях и обычно всегда равна единице. Давайте познакомимся с использованием данного класса на практике. Попробуем изменить интенсивность каждой из трех составляющих цвета на некоторую величину и посмотрим на результат — проект `ColorMatrixSample` (листинг 8.8).

Листинг 8.8. Преобразование картинки с помощью матрицы

```
'--------------------------------------
' Преобразование картинки © 2004 Александр Климов
'--------------------------------------
Imports System.Drawing.Drawing2D
Imports System.Drawing.Imaging

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
```

```
        Dim img As Image = PictureBox1.Image

        Dim copy As Bitmap = New Bitmap(img.Width, img.Height)

        Dim ia As ImageAttributes = New ImageAttributes
        Dim cm As ColorMatrix = New ColorMatrix
        cm.Matrix00 = cm.Matrix11 = cm.Matrix22 = 0.99F

        ia.SetColorMatrix(cm)

        Dim g As Graphics
        g = Graphics.FromImage(copy)
        g.DrawImage(img, New Rectangle(0, 0, img.Width, img.Height), 0,
0, img.Width, img.Height, GraphicsUnit.Pixel, ia)
        PictureBox1.Image = copy
        g.Dispose()
        img.Dispose()
    End Sub
```

В этом примере я взял картинку Autumn.jpg с изображением осеннего парка из папки Windows\Web\Wallpaper операционной системы Windows XP. Все листья на деревьях оригинала, как и положено, желтого цвета. После применения матрицы с заданными установками все листья стали сочного зеленого цвета, и осенний парк буквально на глазах преобразился в летний! К сожалению, на бумаге нельзя воспроизвести данный эффект, поэтому настоятельно рекомендую запустить проект и посмотреть на преображенную картинку.

Для закрепления материала продолжим наши опыты. Мы недавно рассматривали эффект преобразования картинки в серый цвет путем прохождения в цикле через каждый пиксел изображения. Эта же задача реализуема и через класс `ColorMatrix` по знакомой уже нам формуле (листинг 8.9).

Листинг 8.9. Преобразование картинки в серый цвет

```
    Private Sub butGray_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles butGray.Click
        ' Преобразование картинки в серый цвет
        Dim img As Image = PictureBox1.Image

        Dim copy As Bitmap = New Bitmap(img.Width, img.Height)
```

```
        Dim ia As ImageAttributes = New ImageAttributes

        ' Алгоритм преобразования в серый цвет
        Dim cm As ColorMatrix = New ColorMatrix(New Single()() _
               {New Single() {0.3, 0.3, 0.3, 0, 0}, _
                New Single() {0.59, 0.59, 0.59, 0, 0}, _
                New Single() {0.11, 0.11, 0.11, 0, 0}, _
                New Single() {0, 0, 0, 1, 0}, _
                New Single() {0, 0, 0, 0, 1}})

        ia.SetColorMatrix(cm)

        Dim g As Graphics
        g = Graphics.FromImage(copy)
        g.DrawImage(img, _
           New Rectangle(0, 0, img.Width, img.Height), 0, 0, _
           img.Width, img.Height, GraphicsUnit.Pixel, ia)
        PictureBox1.Image = copy
        g.Dispose()
        img.Dispose()
    End Sub
```

Еще одной распространенной операцией при работе с изображениями является получение негатива. В этом случае следует изменить значения основных цветов на противоположные (листинг 8.10).

Листинг 8.10. Получение негатива

```
    Private Sub butNegative_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles butNegative.Click

        ' Получение негатива
        Dim img As Image = PictureBox1.Image
        Dim copy As Bitmap = New Bitmap(img.Width, img.Height)
        Dim ia As ImageAttributes = New ImageAttributes

        ' Алгоритм преобразования в негатив
        Dim cm As ColorMatrix = New ColorMatrix(New Single()() _
               {New Single() {-1, 0, 0, 0, 0}, _
                New Single() {0, -1, 0, 0, 0}, _
```

```
            New Single() {0, 0, -1, 0, 0}, _
            New Single() {0, 0, 0, 1, 0}, _
            New Single() {0, 0, 0, 0, 1}})

        ia.SetColorMatrix(cm)

        Dim g As Graphics
        g = Graphics.FromImage(copy)
        g.DrawImage(img, _
           New Rectangle(0, 0, img.Width, img.Height), 0, 0, _
           img.Width, img.Height, GraphicsUnit.Pixel, ia)
        PictureBox1.Image = copy
        g.Dispose()
        img.Dispose()
    End Sub
```

Ну, и последний пример, который мы с вами разберем, — получение полупрозрачности. В этом случае нужно задействовать элемент матрицы `Matrix33`. Установив его в значение 0.5, мы тем самым даем понять, что нам требуется преобразовать компонент `Alpha`, отвечающий за прозрачность (листинг 8.11).

Листинг 8.11. Создание полупрозрачной картинки

```
    Private Sub butHalfTrans_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles butHalfTrans.Click

        ' Получение полупрозрачной картинки
        Dim img As Image = PictureBox1.Image
        Dim copy As Bitmap = New Bitmap(img.Width, img.Height)
        Dim ia As ImageAttributes = New ImageAttributes

        ' Алгоритм получения полупрозрачности
        Dim cm As ColorMatrix = New ColorMatrix(New Single()() _
               {New Single() {1, 0, 0, 0, 0}, _
                New Single() {0, 1, 0, 0, 0}, _
                New Single() {0, 0, 1, 0, 0}, _
                New Single() {0, 0, 0, 0.5, 0}, _
                New Single() {0, 0, 0, 0, 1}})
```

```
        ia.SetColorMatrix(cm)

        Dim g As Graphics
        g = Graphics.FromImage(copy)
        g.DrawImage(img, _
           New Rectangle(0, 0, img.Width, img.Height), 0, 0, _
           img.Width, img.Height, GraphicsUnit.Pixel, ia)
        PictureBox1.Image = copy
        g.Dispose()
        img.Dispose()
    End Sub
```

Рассмотрев все эти примеры, хочется создать какой-нибудь интересный эффект. Я предлагаю создать пример плавного проявления одной картинки из другой. Приступим.

8.3.2. Эффект плавного проявления одной картинки из другой

Идея, лежащая в основе эффекта проявления картинки, очень проста. Мы уже знаем, как можно управлять прозрачностью картинки. Таким образом, мы можем через определенные промежутки времени изменять уровень прозрачности одной картинки, давая возможность появления другой картинки, которая находится на этом же участке формы. Для усиления эффекта можно также добавить возможность случайного изменения цветовых составляющих матрицы `ColorMatrix` — проект `ColorMatrixFX` (листинг 8.12).

Листинг 8.12. Проявление картинки

```
'--------------------------------------
' Проявление картинки © 2004 Александр Климов
'--------------------------------------
Imports System.Drawing.Drawing2D
Imports System.Drawing.Imaging

    ' Уровень прозрачности
    Private transparent As Single = 0.1F

    ' Шаг изменения прозрачности
    Private trStep As Single = 0.1F
```

```
    ' Массив матрицы цветов
    Private apts(,) As Single = {{1.0F, 0.0F, 0.0F, 0.0F, 0.0F}, _
                                 {0.0F, 1.0F, 0.0F, 0.0F, 0.0F}, _
                                 {0.0F, 0.0F, 1.0F, 0.0F, 0.0F}, _
                                 {0.0F, 0.0F, 0.0F, transparent, 0.0F}, _
                                 {0.0F, 0.0F, 0.0F, 0.0F, 1.0F}}

    Private backImg As Image
    Private frontImg As Image

    Private cm As New ColorMatrix
    Private ia As New ImageAttributes
    Private r As New Rectangle
    Private rnd As New Random

    Private Sub Form1_Paint(ByVal sender As Object, ByVal e As
System.Windows.Forms.PaintEventArgs) Handles MyBase.Paint
        DrawFX(e, transparent)
    End Sub

    Private Sub Form1_Load(ByVal sender As Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        ' Для уменьшения мерцания формы
        MyClass.SetStyle(ControlStyles.UserPaint, True)
        MyClass.SetStyle(ControlStyles.AllPaintingInWmPaint, True)
        MyClass.SetStyle(ControlStyles.DoubleBuffer, True)

        backImg = PictureBox1.Image
        frontImg = PictureBox2.Image
        r = New Rectangle(0, 0, MyClass.ClientSize.Width, _
                          ClientSize.Height)

        Dim i, j As Integer
        For i = 0 To 4
            For j = 0 To 4
                cm.Item(i, j) = apts(i, j)
            Next
        Next
    End Sub
```

```
    Private Sub Timer1_Tick(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Timer1.Tick
        If transparent < 0 Or transparent > 1 Then
            trStep *= -1
            cm.Matrix01 = Convert.ToSingle(rnd.NextDouble)
            cm.Matrix12 = Convert.ToSingle(rnd.NextDouble)
            cm.Matrix23 = Convert.ToSingle(rnd.NextDouble)
        End If

        transparent += trStep
        cm.Matrix33 = transparent
        Refresh()
    End Sub

    Private Sub DrawFX(ByVal e As PaintEventArgs, ByVal trsp As Single)
        Dim g As Graphics = e.Graphics
        g.DrawImage(backImg, r)

        ia.SetColorMatrix(cm)
        g.DrawImage(frontImg, r, 0, 0, _
          frontImg.Width, frontImg.Height, _
          GraphicsUnit.Pixel, ia)
    End Sub
```

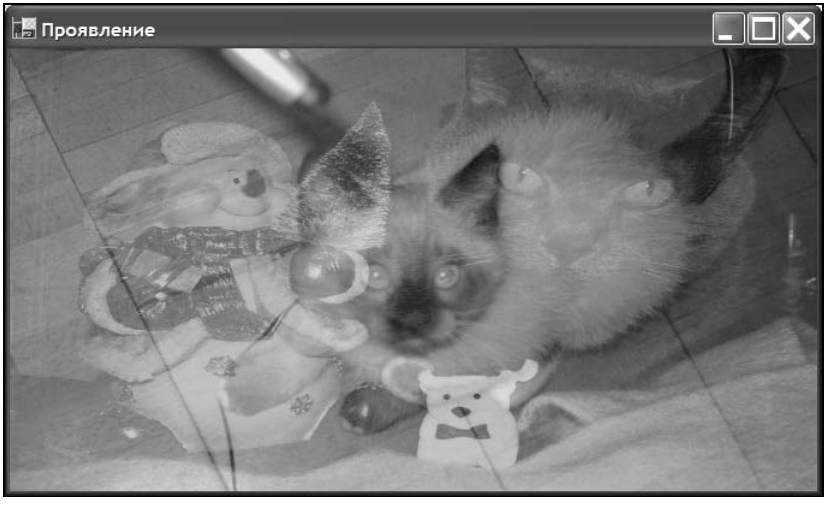

Рис. 8.6. Проявление картинки

Я думаю, что вы уже в состоянии самостоятельно разобраться в примере. Основной код сосредоточен в событии таймера `Tick`, который к тому же вызывает написанную процедуру `DrawFX`. Я постарался поймать один из моментов проявления одной картинки из другой (рис. 8.6). Оригиналы картинок находятся на прилагаемом компакт-диске.

8.4. Метод *DrawImage*

Метод `DrawImage` является очень мощным графическим методом Visual Basic .NET 2003. Для ветеранов старого Visual Basic 6.0 скажу, что слабым его аналогом является метод `PaintPicture`. Достаточно только сказать, что метод `DrawImage` имеет 30 перегруженных версий. Одно только их описание займет не одну страницу текста. Мы рассмотрим только самые интересные случаи. Одна из версий метода выглядит следующим образом:

```
Overloads Public Sub DrawImage( _
   ByVal image As Image, _
   ByVal destPoints() As Point _
)
```

В качестве параметров в данной версии используется массив из трех точек — вершин графического объекта. Эти точки отвечают за верхний левый, верхний правый и нижний левый углы картинки. Таким образом, по этим заданным точкам можно создать фигуру в виде параллелограмма.

8.4.1. Параллелограмм

Код для построения параллелограмма очень простой и короткий. Сначала создадим новый проект `Warp`. Весь код будет выполняться при прорисовке формы (листинг 8.13).

Листинг 8.13. Создание параллелограмма

```
'--------------------------------------
' Изображение - параллелограмм © 2004 Александр Климов
'--------------------------------------
    Private Sub Form1_Paint(ByVal sender As Object, ByVal e As
System.Windows.Forms.PaintEventArgs) Handles MyBase.Paint
        ' Загрузим картинку из файла
        Dim img As Image = Image.FromFile(Application.StartupPath() _
                        & "..\..\cat2.jpg")

        ' Создадим массив точек,
        ' определяющих углы картинки
```

```
        Dim w As Single = img.Width
        Dim h As Single = img.Height
        Dim corners As Point() = { _
                                   New Point(w * 0.5, 0), _
                                   New Point(w, h * 0.5), _
                                   New Point(0, h * 0.5)}

        ' Выводим изображение на экран
        e.Graphics.DrawImage(img, corners)
    End Sub
```

Разберем код. Сначала мы загружаем в переменную `img` картинку из имеющегося файла. Затем создаем массив из трех точек. Для удобства я воспользовался размерами самой картинки, проделал с ними небольшие манипуляции и в результате получил точки, соответствующие середине картинки по горизонтали и вертикали, затем присвоил полученные координаты трем точкам. А дальше все просто. Вызываем метод `DrawImage` с установленными новыми координатами и выводим деформированную картинку на экран (рис. 8.7).

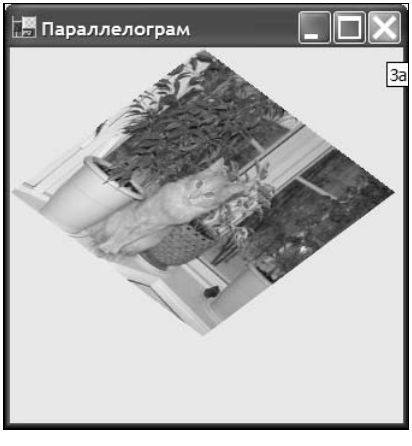

Рис. 8.7. Параллелограмм

8.4.2. Буксировка картинки

Идею следующего примера, который мы сейчас рассмотрим, я подглядел на сайте Рода Стивенса (Rod Stephens) **http://www.vb-helper.com/**, на котором вы найдете много интересных примеров. Кстати, Род Стивенс является автором нескольких книг, которые были опубликованы на русском языке.

Приведенный только что пример по созданию параллелограмма был слишком прост, и рассказал я о нем исключительно для ознакомления. Но на основе этого примера можно добавить в программу больше интерактивности. Например, позволить пользователю воздействовать на картинку мышью. Раз мы увидели, что можно создавать деформации картинки посредством трех точек, то можно попробовать менять координаты этих точек динамически. Для наглядности я создал специальные метки в виде красных окружностей. Вы можете подвести курсор мыши к любой из этих меток и перетащить ее в другое место. В результате программа пересчитает новые координаты выбранной точки и динамически изменит картинку. Не советую использовать для примеров картинки слишком больших размеров. Где-то за кулисами программы идет обработка слишком больших данных, и ваша картинка будет меняться очень медленно. Давайте перейдем непосредственно к примеру. Создадим новый проект `Drag` и расположим на форме два элемента `PictureBox`: `picSource` и `picDest`. В элементе `picSource` у нас будет содержаться картинка-источник. Загрузите в него любую вашу картинку (свойство `Image`) и сделайте его невидимым (свойству `Visible` присвойте значение `False`). С визуальным программированием закончено. Откройте редактор кода и напишите следующий код (листинг 8.14).

Листинг 8.14. Динамическое воздействие на картинку

```
'--------------------------------------
' Буксировка картинки © 2004 Александр Климов
' Идея Рода Стивенса http://www.vb-helper.com
'--------------------------------------
    ' Радиус окружностей для меток
    Private Const CORNER_RADIUS As Integer = 7
    Private BmSource As Bitmap
    Private BmDest As Bitmap
    Private Corners As Point()
    Private DragCorner As Long

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        BmSource = New Bitmap(picSource.Image)
        BmDest = New Bitmap( _
                         CInt(BmSource.Width), _
                         CInt(BmSource.Height))

        Corners = New Point() { _
                           New Point(0, 0), _
```

```
                                New Point(BmSource.Width, 0), _
                                New Point(0, BmSource.Height)}

        DragCorner = -1

        ' Выводим картинку.
        WarpImage()
    End Sub

    ' Рисуем картинку для опытов.
    Private Sub WarpImage()
        ' Создадим объект Graphics для картинки.
        Dim gr_dest As Graphics = Graphics.FromImage(BmDest)

        ' Копируем картинку-источник в картинку-приемник.
        gr_dest.Clear(picDest.BackColor)
        gr_dest.DrawImage(BmSource, Corners)

        ' Рисуем три метки-окружности.
        Dim i As Long
        For i = 0 To 2
            Dim pn As New Pen(Color.Red, 3)
            gr_dest.DrawEllipse(pn, _
                Corners(i).X - CORNER_RADIUS, _
                Corners(i).Y - CORNER_RADIUS, _
                2 * CORNER_RADIUS, 2 * CORNER_RADIUS)
        Next i

        ' Показываем результат.
        picDest.Image = BmDest
    End Sub

    ' Начинаем перетаскивание точек.
    Private Sub picDest_MouseDown(ByVal sender As Object, ByVal e As
System.Windows.Forms.MouseEventArgs) Handles picDest.MouseDown
        Dim i As Long

        ' Если мышь рядом с одной из меток.
        For i = 0 To 2
```

```
            If (Math.Abs(Corners(i).X - e.X) < CORNER_RADIUS) And _
               (Math.Abs(Corners(i).Y - e.Y) < CORNER_RADIUS) _
            Then
                ' Начинаем перетаскивать угол.
                DragCorner = i
                Exit For
            End If
        Next i
    End Sub

    Private Sub picDest_MouseMove(ByVal sender As Object, ByVal e As
System.Windows.Forms.MouseEventArgs) Handles picDest.MouseMove
        ' Проверка, перетаскиваем ли угол картинки.
        If DragCorner < 0 Then Exit Sub

        Corners(DragCorner).X = e.X
        If Corners(DragCorner).X < 0 Then
            Corners(DragCorner).X = 0
        ElseIf Corners(DragCorner).X > BmDest.Width Then
            Corners(DragCorner).X = BmDest.Width
        End If
        Corners(DragCorner).Y = e.Y
        If Corners(DragCorner).Y < 0 Then
            Corners(DragCorner).Y = 0
        ElseIf Corners(DragCorner).Y > BmDest.Height Then
            Corners(DragCorner).Y = BmDest.Height
        End If

        WarpImage()
    End Sub

    ' Конец буксировке.
    Private Sub picDest_MouseUp(ByVal sender As Object, ByVal e As
System.Windows.Forms.MouseEventArgs) Handles picDest.MouseUp
        DragCorner = -1
    End Sub
```

Разберем код. Вначале я объявил несколько глобальных переменных. В переменной `CORNER_RADIUS` содержится значение радиуса для окружности ме-

ток. С помощью этих меток мы будем деформировать картинку. Далее идут две переменные BmSource и BmDest, в которых будут содержаться графические объекты (картинка-источник и картинка-приемник). Переменная Corners является массивом точек, а переменная DragCorner — индикатором возможности перетаскивания угла картинки. Во время загрузки формы я присваиваю объявленным ранее переменным BmSource и BmDes предварительно загруженную картинку и определяю массив из трех точек, которые будут использоваться в нашей программе. Для примера я взял координаты верхнего левого, верхнего правого и нижнего левого углов картинки. За деформацию картинки будет отвечать процедура WrapImage. Давайте разберем эту процедуру в деталях. Сначала мы создаем объект Graphics для картинки и копируем в нее картинку-источник. Чтобы нам было легче ориентироваться, мы рисуем дополнительно три метки-окружности:

```
' Рисуем три метки-окружности.
For i = 0 To 2
    Dim pn As New Pen(Color.Red, 3)
    gr_dest.DrawEllipse(pn, _
        Corners(i).X - CORNER_RADIUS, _
        Corners(i).Y - CORNER_RADIUS, _
        2 * CORNER_RADIUS, 2 * CORNER_RADIUS)
Next i
```

И выводим результат в элемент picDest. Самое интересное содержится в событиях picDest_MouseDown и picDest_MouseMove. При нажатии кнопки мыши мы определяем координаты мыши, и если в этот момент мышь находилась внутри окружностей-меток, то начинаем буксировку угла картинки. Сама буксировка происходит в обработчике события picDest_MouseMove. В нем мы динамически переопределяем углы картинки, передавая им текущие координаты мыши.

8.4.3. Куб с картинками на гранях

Особо наблюдательные читатели могли увидеть из примера с буксировкой картинки один интересный момент. Если вы представляете себе, как рисуют объемный куб на плоском экране, то без труда могли узнать в параллелограммах элементы куба — его грани. Таким образом, можно легко нарисовать куб с заданными картинками. Такой графический куб очень эффектно смотрится на мониторе. Давайте реализуем эту задачу в новом проекте DrawCube. Для начала нам понадобятся три графических файла, которые будут загружаться в форму во время запуска программы. Найдите в своем семейном альбоме три наиболее красивые фотографии и запомните их расположение на вашем компьютере. В моем примере я отобрал три фотографии

своего кота, назвал их 1.jpg, 2.jpg, 3.jpg и поместил в папку bin. Сделано это только для удобства демонстрации учебного примера, в своих проектах вы можете изменить путь к своим отобранным фотографиям. Для вывода куба на экран будем использовать событие `Click` кнопки `Button` (листинг 8.15).

Листинг 8.15. Создание куба с картинками на его гранях

```
'--------------------------------------
' Куб с картинками © 2004 Александр Климов
'--------------------------------------
    Private Sub butDrawCube_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles butDrawCube.Click
        ' Создадим объект Graphics для куба с картинками
        Dim g As Graphics
        g = CreateGraphics()
        g.TranslateTransform(100, 200)

        Dim m_Images(2) As Image
        Dim app_path As String
        app_path = Application.StartupPath
        m_Images(0) = Image.FromFile(app_path & "\1.jpg")
        m_Images(1) = Image.FromFile(app_path & "\2.jpg")
        m_Images(2) = Image.FromFile(app_path & "\3.jpg")

        ' Три точки, определяющих переднюю грань куба
        Dim face1(2) As Point
        face1(0) = New Point(-50, -20)
        face1(1) = New Point(250, -20)
        face1(2) = New Point(-50, 230)

        ' Верхняя грань куба
        Dim face2(2) As Point
        face2(0) = New Point(70, -140)
        face2(1) = New Point(370, -140)
        face2(2) = New Point(-50, -20)

        ' Боковая грань куба
        Dim face3(2) As Point
```

```
        face3(0) = New Point(250, -20)
        face3(1) = New Point(370, -140)
        face3(2) = New Point(250, 230)

        ' Выводим картинки на экран
        g.DrawImage(m_Images(1), face1)
        g.DrawImage(m_Images(2), face2)
        g.DrawImage(m_Images(0), face3)
        g.Dispose()
    End Sub
```

Запустите проект и полюбуйтесь на получившийся результат (рис. 8.8). Не правда ли, красиво? Сначала мы загружаем фотографии в массив `m_Images`, затем создаем три грани куба. Каждая грань куба является параллелограммом и может быть определена с помощью массива из трех точек. Главное — это совместить параллелограммы таким образом, чтобы на экране появился куб.

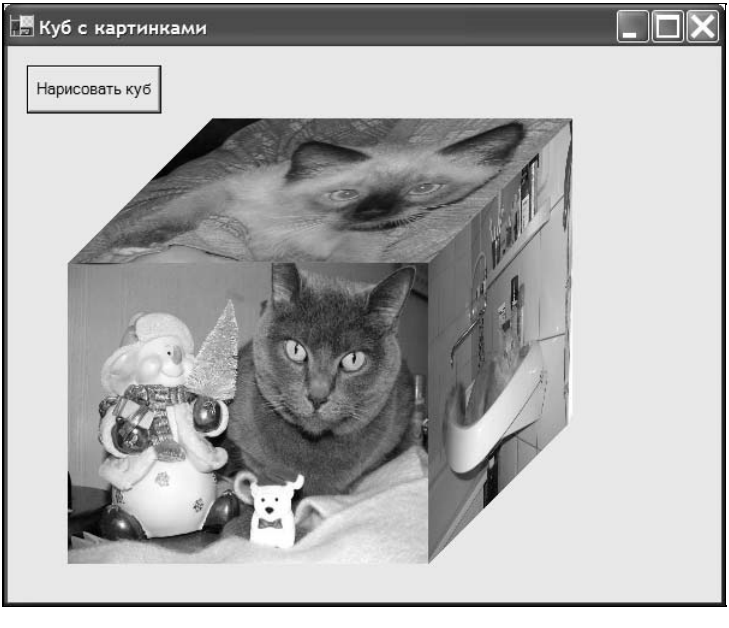

Рис. 8.8. Куб с картинками

8.4.4. Поворот картинки на заданный угол

Если нужно повернуть картинку на заданный угол, то используем ту же технику, рассмотренную в предыдущих примерах. Наша задача — рассчитать по тригонометрическим формулам вершины картинки после смещения на выбранный угол и вывести результат на экран. Эту задачу выполняет пример из проекта `RotateAngle`. Разместим на форме два элемента `PictureBox`, надпись `Label`, текстовое поле `TextBox` и кнопку `Button`. Загрузите в картинку `picFrom` любое небольшое изображение, которое будет объектом наших исследований. Весь основной код выполняется при нажатии на кнопку **butRotate** (листинг 8.16).

Листинг 8.16. Поворот картинки на заданный угол

```
'-----------------------------------------------
' Поворот на заданный угол © 2004 Александр Климов
'-----------------------------------------------
Imports System.Math

    Private Sub butRotate_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles butRotate.Click
        ' Получим доступ к картинке из объекта picFrom.
        Dim bm_from As New Bitmap(picFrom.Image)

        ' Создадим массив точек для углов картинки.
        Dim wid As Single = bm_from.Width
        Dim hgt As Single = bm_from.Height
        Dim corners As Point() = { _
            New Point(0, 0), _
            New Point(wid, 0), _
            New Point(0, hgt), _
            New Point(wid, hgt)}

        ' Находим центр картинки.
        Dim cx As Single = wid / 2
        Dim cy As Single = hgt / 2
        Dim i As Long
        For i = 0 To 3
            corners(i).X -= cx
```

```
    corners(i).Y -= cy
Next i

' Начинаем вращать.
Dim theta As Single = Single.Parse(txtAngle.Text) * PI / 180.0
Dim sin_theta As Single = Sin(theta)
Dim cos_theta As Single = Cos(theta)
Dim X As Single
Dim Y As Single
For i = 0 To 3
    X = corners(i).X
    Y = corners(i).Y
    corners(i).X = X * cos_theta + Y * sin_theta
    corners(i).Y = -X * sin_theta + Y * cos_theta
Next i

Dim xmin As Single = corners(0).X
Dim ymin As Single = corners(0).Y
For i = 1 To 3
    If xmin > corners(i).X Then xmin = corners(i).X
    If ymin > corners(i).Y Then ymin = corners(i).Y
Next i
For i = 0 To 3
    corners(i).X -= xmin
    corners(i).Y -= ymin
Next i

' Создаем новые объекты Bitmap и Graphics.
Dim bm_to As New Bitmap(CInt(-2 * xmin), CInt(-2 * ymin))
Dim g_to As Graphics = Graphics.FromImage(bm_to)

' Так как метод DrawImage использует три точки,
' то переопределяем массив, удалив последнюю точку угла.
ReDim Preserve corners(2)

' Рисуем полученное изображение в picTo
' и выводим результат на экран.
g_to.DrawImage(bm_from, corners)
```

```
        picTo.Image = bm_to
    End Sub
```

Теперь можете запустить проект и вводить в текстовое поле любое числовое значение. Для положительных значений поворот осуществляется против часовой стрелки. Для поворота картинки по часовой стрелке используйте отрицательные числа.

8.4.5. Вращение картинки с помощью таймера

Если мы с вами научились поворачивать картинку на любой угол, то готовы приступить к разбору примера вращения картинки. Создаем новый проект `Spin`. Аналогично предыдущему примеру разместим на форме кнопку `btnStart` с надписью **Старт**, а также два графических объекта `PictureBox`. Одному из них присвоим имя `picFrom` (он будет содержать картинку-источник), другому — `picTo` (здесь будет выполняться работа над картинкой). Присвойте обоим в свойстве `SizeMode` значение `AutoSize`. Загрузите в `picFrom` какую-нибудь картинку. В своем примере я использовал картинку кота, сидящего в раковине умывальника. Также добавляем в проект таймер `Timer1`. Измените интервал таймера на 1. Предварительные приготовления закончены. Теперь приступаем к кодированию. Для начала объявим глобальные переменные (листинг 8.17).

Листинг 8.17. Вращение картинки

```
'---------------------------------------
' Вращение картинки © 2004 Александр Климов
'---------------------------------------
    ' Для копирования картинки из источника
    Private bmp_from As Bitmap
    Private fromWid As Integer
    Private fromHgt As Integer
    Private fromCx As Single
    Private fromCy As Single
    Private fromCorners As PointF()

    ' Для вращаемой картинки
    Private bmp_to As Bitmap
    Private toWid As Integer
    Private toHgt As Integer
```

```
    Private toCx As Single
    Private toCy As Single

    Private FramesDrawn As Long
    Private ToBackColor As Color

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        bmp_from = New Bitmap(picFrom.Image)
        fromWid = bmp_from.Width
        fromHgt = bmp_from.Height
        fromCx = fromWid / 2
        fromCy = fromHgt / 2
        fromCorners = New PointF() { _
            New PointF(0, 0), _
            New PointF(fromWid, 0), _
            New PointF(0, fromHgt), _
            New PointF(fromWid, fromHgt)}

        toWid = Math.Sqrt(fromWid * fromWid + fromHgt * fromHgt)
        toHgt = toWid
        toCx = toWid / 2
        toCy = toHgt / 2
        bmp_to = New Bitmap(toWid, toHgt)

        Dim i As Long
        For i = 0 To 3
            fromCorners(i).X -= fromCx
            fromCorners(i).Y -= fromCy
        Next i

        ToBackColor = picTo.BackColor
    End Sub

    Private Sub Timer1_Tick(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Timer1.Tick

        FramesDrawn += 1
```

```
    ' Увеличиваем угол
    Const dtheta = 3 * Math.PI / 180.0
    Static theta As Single = 0
    theta += dtheta

    ' Помещаем в массив координаты углов картинки
    Dim corners() As PointF = fromCorners

    ' Удаляем одну вершину как лишнюю
    ReDim Preserve corners(2)

    ' Вращаем
    Dim sin_theta As Single = Math.Sin(theta)
    Dim cos_theta As Single = Math.Cos(theta)
    Dim X As Single
    Dim Y As Single
    Dim i As Long
    For i = 0 To 2
        X = corners(i).X
        Y = corners(i).Y
        corners(i).X = X * cos_theta + Y * sin_theta
        corners(i).Y = -X * sin_theta + Y * cos_theta
    Next i

    For i = 0 To 2
        corners(i).X += toCx
        corners(i).Y += toCy
    Next i

    ' Создаем новую картинку и объект Graphics.
    Dim g_to As Graphics = Graphics.FromImage(bmp_to)

    ' Выводим результат на экран.
    g_to.Clear(ToBackColor)
    g_to.DrawImage(bmp_from, corners)

    picTo.Image = bmp_to
End Sub
```

```
    Private Sub btnStart_Click(ByVal sender As Object, ByVal e As
System.EventArgs) Handles btnStart.Click
        ' Включаем или останавливаем вращение.
        Static start_time As DateTime

        If Timer1.Enabled Then
            ' Останавливаем вращение
            Dim stop_time As DateTime
            stop_time = Now
            Timer1.Enabled = False
            btnStart.Text = "Старт"

        Else
            ' Начинаем вращение
            Timer1.Enabled = True
            btnStart.Text = "Стоп"
            FramesDrawn = 0
            start_time = Now
        End If
    End Sub
```

При загрузке формы происходит инициализация переменных, содержащих размеры картинки, координаты ее центра и вершин. Само вращение картинки выполняется в событии таймера, который включается или выключается кнопкой. Кстати, попробуйте закомментировать строчки кода, отвечающие за цвет фона графического поля. В этом случае получится весьма интересный эффект.

8.5. Сохранение картинки

В VB 6.0 можно было сохранять рисунки только в неудобном BMP-формате, хотя он и позволял работать с картинками других форматов. Приходилось использовать очень сложные программы от других фирм. Теперь удивительно легко можно конвертировать один графический формат в другой. Для этого достаточно воспользоваться классом `ImageFormat` с указанием нужного формата. Поддерживаются все популярные форматы: BMP, GIF, JPG, TIFF, ICO и еще ряд других. В проекте `SaveImage` (листинг 8.18) демонстрируется пример сохранения картинки из объекта `PictureBox` на диск компьютера в четырех наиболее распространенных форматах (BMP, GIF, JPG и PNG).

Листинг 8.18. Сохранение рисунков на диске

```
    Private Sub butSave_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles butPng.Click
        Dim file_path As String = Application.ExecutablePath
        file_path = file_path.Substring(0, _
             file_path.LastIndexOf("\bin")) & _
            "\test."

        ' Используем картинку из графического поля
        Dim pic As Bitmap = picTest.Image

        ' Сохраняем картинки в разных форматах.
        pic.Save(file_path & "bmp", ImageFormat.Bmp)
        pic.Save(file_path & "jpg", ImageFormat.Jpeg)
        pic.Save(file_path & "gif", ImageFormat.Gif)
        pic.Save(file_path & "png", ImageFormat.Png)
    End Sub
```

Не забудьте в начале проекта импортировать требуемое пространство имен:

```
Imports System.Drawing.Imaging
```

Уже только один этот пример показывает, насколько далеко ушел VB. NET 2003 от своего предшественника VB 6.0. Перефразируя одну известную рекламу, можно сказать: Вы все еще программируете на VB 6.0? Тогда мы идем к вам :-).

8.6. Класс *ImageAnimator*

Теперь VB .NET 2003 поддерживает анимированные картинки. Не надо больше устанавливать чужие компоненты. Причем класс `ImageAnimator` позволяет выводить не только анимированную картинку, но и работать с отдельными кадрами анимации. Рассмотрим этот класс поподробнее. Всего у него четыре метода. Например, метод `CanAnimate` позволяет узнать, является ли картинка анимированной. Вот простой пример вывода анимированной картинки на форме — проект `Animator` (листинг 8.19).

Листинг 8.19. Работа с анимированными картинками

```
'--------------------------------------
' Анимированные картинки © 2004 Александр Климов
'--------------------------------------
```

```
Imports System.Drawing

    ' Создадим объект Bitmap из файла.
    Private animatedImage As New Bitmap(Application.StartupPath _
           & "..\..\anicat.gif")
    Private currentlyAnimating As Boolean = False

    ' Метод для анимации.
    Public Sub AnimateImage()
        If Not currentlyAnimating Then
            ' Начинаем анимацию.
            ImageAnimator.Animate(animatedImage, _
            New EventHandler(AddressOf Me.OnFrameChanged))
            currentlyAnimating = True
        End If
    End Sub

    Private Sub OnFrameChanged(ByVal o As Object, ByVal e As EventArgs)
        ' Принудительный вызов события Paint
        Me.Invalidate()
    End Sub

    Protected Overrides Sub OnPaint(ByVal e As PaintEventArgs)
        ' Начинаем анимацию.
        AnimateImage()

        ' Получим следующий кадр.
        ImageAnimator.UpdateFrames()

        ' Выводим кадр на экран.
        e.Graphics.DrawImage(Me.animatedImage, New Point(130, 30))
    End Sub

    Private Sub butStop_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles butStop.Click
        ' Останавливаем анимацию
        ImageAnimator.StopAnimate(animatedImage, _
            New EventHandler(AddressOf Me.OnFrameChanged))
    End Sub
```

Сначала я объявил объект `Bitmap`, в который загрузил анимированную картинку из файла anicat.gif (в случае необходимости измените путь к файлу). Затем создал метод для анимации `AnimateImage`, в котором происходит вызов метода `Animate`.

8.6.1. Кадры анимации

Как уже говорилось, класс `ImageAnimator` позволяет не только воспроизводить анимированную картинку, но и получать отдельные кадры анимации. Посмотрим, как это реализуется в проекте `Animator2`. Для демонстрации нам понадобятся три графических поля и одна кнопка. Анимированная картинка anicat.gif находится на диске в папке с проектом. В случае необходимости измените путь к собственному файлу в коде программы. Программа извлекает из анимированной картинки три кадра — первый, третий и последний и выводит их в трех графических полях. Вот код для кнопки, при нажатии на которую и происходит вывод кадров на экран (листинг 8.20).

Листинг 8.20. Извлечение отдельных кадров анимации

```
'--------------------------------------
' Кадры анимации © 2004 Александр Климов
'--------------------------------------
Imports System.Drawing.Drawing2D
Imports System.Drawing


    Private Sub butGetFrames_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles butGetFrames.Click
        ' Загружаем картинку и извлекаем кадры из нее
        Me.Cursor = Cursors.WaitCursor
        Dim anim As New Animation(Application.StartupPath _
          & "..\..\anicat.gif")
        Me.Cursor = Cursors.Default

        ' Выводим первый кадр картинки
        PictureBox1.Image = anim.Frame(0)
        ' Выводим третий кадр картинки
        PictureBox2.Image = anim.Frame(2)
        ' Выводим последний кадр картинки
        PictureBox3.Image = anim.Frame(anim.FrameCount - 1)
    End Sub
```

Как вы видите, в коде осуществляется обращение к новому классу `Animation`, в котором происходит основная работа по извлечению кадров анимации (листинг 8.21).

Листинг 8.21. Создание класса для анимации

```
Class Animation
    Implements IDisposable

    Dim m_img As System.Drawing.Image
    Dim m_Frames As New System.Collections.ArrayList
    Dim m_FrameCount As Integer

    Sub New(ByVal imgPath As String)
        ' Загружаем картинку
        m_img = System.Drawing.Image.FromFile(imgPath)

        ' Обработка возможной ошибки
        If ImageAnimator.CanAnimate(m_img) = False Then
            m_img.Dispose()
            Throw New ArgumentException("Это не анимированная картинка")
        End If

        ' Число кадров
        Dim frame_count As New
        Imaging.FrameDimension(m_img.FrameDimensionsList _
           (0))
        m_FrameCount = m_img.GetFrameCount(frame_count)

        ' Извлекаем все кадры анимации
        AddCurrentFrame()
        ImageAnimator.Animate(m_img, New EventHandler(AddressOf _
            OnFrameChanged))

        Do
        Loop Until GetAll
    End Sub

    ' Возвращает общее число кадров
    Public ReadOnly Property FrameCount() As Integer
```

```
        Get
            Return m_FrameCount
        End Get
    End Property

    ' Возвращает картинку кадра, хранимую в массиве
    Public ReadOnly Property Frame(ByVal index As Integer) As Bitmap
        Get
            Return DirectCast(m_Frames(index), Bitmap)
        End Get
    End Property

    Public Sub Dispose() Implements System.IDisposable.Dispose
        StopAnimate()
    End Sub

    ' Метод, возникающий при смене кадра
    Private Sub OnFrameChanged(ByVal sender As Object, ByVal e As
EventArgs)
        ' Останавливаемся при достижении последнего кадра
        If GetAll Then
            StopAnimate()
            Exit Sub
        End If

        ' Сменяем кадр и сохраняем его в массиве
        ImageAnimator.UpdateFrames(m_img)
        AddCurrentFrame()
    End Sub

    ' Процедура добавления текущего кадра
    ' в массив
    Private Sub AddCurrentFrame()
        ' Создаем объект Bitmap
        Dim bmp As New Bitmap(m_img.Width, m_img.Height)
        ' Создаем объект Graphics
        Dim g As Graphics = Graphics.FromImage _
           (DirectCast(bmp, System.Drawing.Image))
        g.DrawImage(m_img, 0, 0)
        ' Добавляем кадр в массив
```

```
        m_Frames.Add(bmp)
    End Sub

    ' Возвращает True при извлечении всех кадров
    Private ReadOnly Property GetAll() As Boolean
        Get
            Return m_Frames.Count = FrameCount
        End Get
    End Property

    ' Останавливаем анимацию и освобождаем все ресурсы
    Private Sub StopAnimate()
        ImageAnimator.StopAnimate(m_img, _
            New EventHandler(AddressOf OnFrameChanged))
        m_img.Dispose()
    End Sub
End Class
```

Все необходимые пояснения даны в комментариях. Запустите проект и нажмите на кнопку. На трех графических полях должны появиться три различных кадра из анимированной картинки (рис. 8.9).

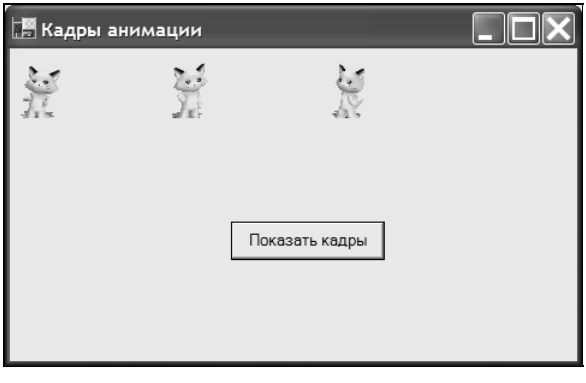

Рис. 8.9. Извлечение кадров из картинки

8.7. Советы и рекомендации

Графические возможности Visual Basic .NET настолько огромны, что невозможно в нескольких главах описать все возможные приемы и трюки. Ищите дополнительную информацию в других книгах и на сайтах Интернета.

ЧАСТЬ III

Клавиатура и мышь

Глава 9

Клавиатура

Клавиатура является одной из основных составляющих компьютера. Даже такая привычная для нас мышь имеет меньший приоритет для системы. Вы можете отключить мышь, и компьютер будет по-прежнему нормально работать. А попробуйте отключить клавиатуру — у вас появится сообщение об ошибке и система откажется загружаться. В этой главе мы разберем несколько интересных примеров, связанных с работой клавиатуры.

Сначала несколько слов об этом устройстве ввода информации. Клавиатура имеет около сотни различных клавиш в зависимости от модификации. Но, несмотря на наличие разных вариантов исполнения клавиатур, существует некий стандартный набор клавиш. Прежде всего это ряд клавиш с буквами и цифрами, клавиши-модификаторы, функциональные клавиши и цифровой блок (у настольных клавиатур). Кроме того, имеются индикаторы со светодиодами, которые могут загораться и выключаться при нажатии некоторых клавиш. На этом разрешите закончить короткое знакомство с клавиатурой и перейти к примерам.

9.1. Переключение раскладки клавиатуры

В отличие от американцев, нам приходится при общении с компьютером применять две, а то и больше раскладок клавиатур. Даже если вы установите английскую версию Windows, то письма, заметки и статьи вам все равно придется писать на русском языке. Таким образом, у вас периодически возникает потребность переключения раскладки. Как правило, достаточно нажать комбинацию клавиш <Shift>+<Alt> (я привожу стандартный способ, существуют и альтернативные), чтобы переключиться на другой язык. В последнее время получили распространение программы, которые автоматически переключают раскладку, пытаясь определить нужный язык по вводимым словам. Но если эти программы способны программно переключать раскладки, то почему бы и нам не сделать то же самое? Для этого нужно воспользоваться свойством `CurrentInputLanguage` класса `InputLanguage`,

который содержится в пространстве имен `System.Globalization` — проект `KeyboardLayout` (листинг 9.1).

Листинг 9.1. Переключение раскладки клавиатуры

```
'--------------------------------------
' Раскладка клавиатуры © 2004 Александр Климов
'--------------------------------------
Imports System.Globalization

    Private Sub Button1_Click(ByVal sender As System.Object, _
           ByVal e As System.EventArgs) Handles Button1.Click

       'InputLanguage.CurrentInputLanguage = InputLanguage.FromCulture( _
                                              'New CultureInfo("ru-RU"))
        InputLanguage.CurrentInputLanguage = InputLanguage.FromCulture( _
                                               New CultureInfo("en-US"))
    End Sub
```

В коде содержатся два режима переключения раскладки клавиатуры. Если вам необходимо переключаться с американской раскладки на русскую, то снимите комментарии с первой строчки кода и закомментируйте соответственно вторую. Я привел стандартную ситуацию, которая встречается на большинстве компьютеров российских пользователей. Если вы используете иную раскладку (украинскую, белорусскую или китайскую), то поищите названия соответствующих раскладок в документации.

9.2. Цветомузыка на клавиатуре

На большинстве клавиатур имеются светодиоды-индикаторы, которые позволяют пользователю отслеживать состояние клавиш <Caps Lock>, <Num Lock> и <Scroll Lock>. Например, если горит индикатор Caps Lock, значит, клавиша <Caps Lock> активирована и все символы в текстовом поле будут выводиться в верхнем реестре (если при этом удерживать клавишу <Shift>, то символы будут выводиться в нижнем реестре). Но то, что может сделать пользователь, можем сделать и мы. Почему бы нам не устроить цветомузыку на клавиатуре пользователя? Можно программно включать и отключать индикаторы в такт какой-нибудь мелодии. Я покажу, как это сделать в проекте `LockKeys` (листинг 9.2). Для реализации шутки достаточно одной кнопки с надписью **Вкл./Выкл.**

Листинг 9.2. Включение и выключение индикаторов клавиш

```
'--------------------------------------
' Цветомузыка на клавиатуре © 2004 Александр Климов
'--------------------------------------
    ' Объявляем функцию API
    Private Declare Sub keybd_event Lib "user32" _
           (ByVal bVk As Byte, _
            ByVal bScan As Byte, _
            ByVal dwFlags As Integer, _
            ByVal dwExtraInfo As Integer)

    ' Константы для функции
    Const VK_NUMLOCK As Short = &H90S
    Const VK_SCROLL As Short = &H91S
    Const VK_CAPITAL As Short = &H14S
    Const KEYEVENTF_EXTENDEDKEY As Short = &H1S
    Const KEYEVENTF_KEYUP As Short = &H2S

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
        ' Клавиша Num Lock
        ' Эмуляция нажатия на клавишу
        keybd_event(VK_NUMLOCK, &H45S, KEYEVENTF_EXTENDEDKEY Or 0, 0)

        ' Отпускаем клавишу
        keybd_event(VK_NUMLOCK, &H45S, _
            KEYEVENTF_EXTENDEDKEY Or KEYEVENTF_KEYUP, 0)

        ' Клавиша CapsLock
        ' Нажимаем на клавишу
        keybd_event(VK_CAPITAL, &H45S, KEYEVENTF_EXTENDEDKEY Or 0, 0)

        ' Отпускаем клавишу
        keybd_event(VK_CAPITAL, &H45S, _
            KEYEVENTF_EXTENDEDKEY Or KEYEVENTF_KEYUP, 0)

        ' Клавиша ScrollLock
        ' Аналогичные действия
```

```
        keybd_event(VK_SCROLL, &H45S, KEYEVENTF_EXTENDEDKEY Or 0, 0)
        keybd_event(VK_SCROLL, &H45S, _
           KEYEVENTF_EXTENDEDKEY Or KEYEVENTF_KEYUP, 0)
    End Sub
```

Чтобы программно нажать на клавиши-индикаторы, нужно объявить функцию Windows API `keybd_event` с необходимыми константами. В обработчике события `Click` кнопки `Button1` нужно дважды вызвать эту функцию для нажатия и отпускания клавиши. Кстати, повторный вызов функции выключит индикатор, поэтому нет необходимости писать какой-то дополнительный код для этой ситуации.

9.3. Состояние клавиши <Caps Lock>

Мы уже знаем, как можно включить или выключить индикаторы клавиш <Caps Lock>, <Num Lock> или <Scroll Lock>. Но как узнать текущее состояние этих клавиш? Здесь нам на помощь придет функция Windows API `GetKeyboardState`, которая позволяет узнать состояние всех клавиш на клавиатуре — проект `KeyState` (листинг 9.3).

Листинг 9.3. Текущее состояние клавиши <Caps Lock>

```
'--------------------------------------
' Текущее состояние клавиши © 2004 Александр Климов
'--------------------------------------
'Функция WinAPI
    Private Declare Function GetKeyboardState Lib "user32" _
                             (ByRef pbKeyState As Byte) As Integer

    ' Константы
    Const VK_NUMLOCK As Short = &H90S
    Const VK_SCROLL As Short = &H91S
    Const VK_CAPITAL As Short = &H14S

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
        Dim CapsLockState As Boolean
        Dim keys(255) As Byte

        GetKeyboardState(keys(0))
```

```
        CapsLockState = keys(VK_CAPITAL)

        If CapsLockState Then
            TextBox1.Text = "CAPSLOCK"
        Else
            TextBox1.Text = "capslock"
        End If
    End Sub
```

Функция получает снимок состояния всех клавиш на клавиатуре. Если установлен бит `&H1`, то клавиша выключена. Если установлен бит `&H80`, клавиша в настоящее время включена. Таким образом, нужно сопоставить бит нужной нам клавиши для получения текущего состояния клавиши. В данном примере я показал, как узнать текущее состояние клавиши <Caps Lock>.

9.4. Получение снимка экрана или активного окна

Вы, возможно, знаете, что при нажатии на клавишу <Print Screen> в буфер обмена копируется снимок всего экрана. Если при этом удерживать клавишу <Alt>, то в буфер обмена копируется только изображение активного окна. Теперь можно открыть любой графический редактор (например, стандартный Paint, входящий в состав Windows) и вставить из буфера обмена полученное изображение. Иногда в программе требуется получить такое изображение. Можно эмулировать нажатия этих клавиш программно. Проект `PrintScreen` (листинг 9.4) демонстрирует технику нажатия клавиш с помощью методов класса `SendKeys`. Добавьте в проект кнопку и графическое поле, в которое будет помещаться полученный снимок из буфера обмена.

Листинг 9.4. Получение снимка экрана

```
'--------------------------------------
' Получение снимка экрана © 2004 Александр Климов
'--------------------------------------
    Function GetScreenshot() As Image
        Return GetScreenshot(False)
    End Function

    Function GetScreenshot(ByVal activeWindow As Boolean) As Image
        ' Для получения снимка активного окна
```

```
        If activeWindow Then
            SendKeys.SendWait("%{PRTSC}")
        Else
            SendKeys.SendWait("{PRTSC 2}")
        End If

        ' в буфере обмена содержится снимок
        Return DirectCast _
           Clipboard.GetDataObject().GetData(DataFormats.Bitmap), _
           Image)
    End Function

    Private Sub butGetScreenShot_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles butGetScreenShot.Click
        picScreenShot.Image = GetScreenshot()

        ' Для получения снимка активного окна используйте True
        picScreenShot.Image = GetScreenshot(True)
    End Sub
```

Как видите, мы определили две версии функции. Первая функция не имеет параметров и получает снимок всего экрана. Вторая функция использует один параметр, который указывает на необходимость получения только снимка активной формы (как при нажатии клавиши <Alt>). В *главе 18* будет показан более сложный пример получения снимков экрана.

9.5. Считаем на калькуляторе

Раз уж мы затронули методы класса `SendKeys`, то давайте рассмотрим еще один пример. Мы можем посылать нажатия клавиш не только своей программе, но и другим запущенным программам. Предположим, у нас уже запущена стандартная программа Калькулятор из поставки Windows. Мы можем сделать активным окно данной программы и послать желаемую комбинацию нажатий клавиш. В проекте `SendToCalculator` (листинг 9.5) реализован данный пример.

Листинг 9.5. Нажатия клавиш в калькуляторе

```
'--------------------------------------
' Нажатия клавиш © 2004 Александр Климов
'--------------------------------------
```

```
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        ' Делаем активным окно Калькулятора
        AppActivate("Калькулятор")

        ' Нажимаем заданную комбинацию клавиш
        SendKeys.SendWait("2*2{ENTER}")
    End Sub
```

Перед тестированием не забудьте запустить программу Калькулятор, иначе вы получите сообщение об ошибке. Выполнение программы осуществляется при загрузке основной формы. Именно в этот момент калькулятор становится активным, и в нем отображается результат умножения двух чисел 2 на 2 (рис. 9.1).

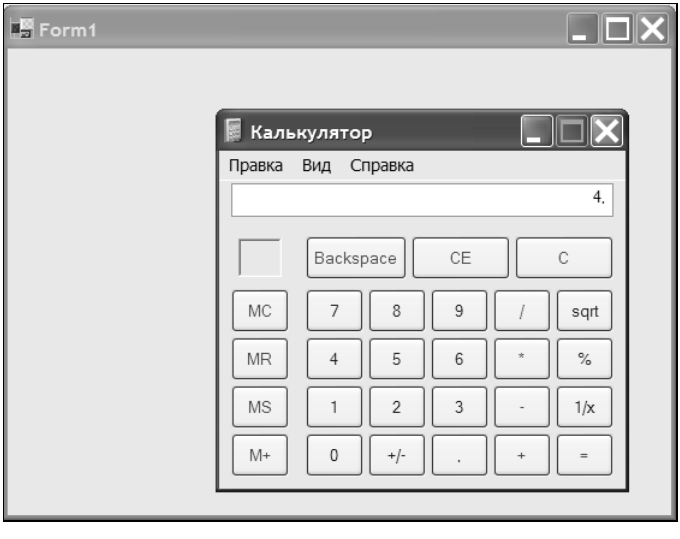

Рис. 9.1. Умножение двух чисел в Калькуляторе

9.6. Советы и рекомендации

- На основе примера 9.3 напишите пример, получающий состояние клавиш <Num Lock> и <Scroll Lock>.
- В примере 9.5 используется уже запущенная программа Калькулятор. Попробуйте усложнить пример. Ваша программа сама должна запустить Калькулятор и послать ей комбинацию клавиш.

Глава 10

Мышь

В этой главе речь пойдет о работе с устройством, которое больше известно нам как мышь. Удивительно, что как только я приступил к написанию главы, ко мне на колени забрался мой кот Рыжик и стал внимательно смотреть на монитор. Как вы, вероятно, знаете, у любой стандартной мыши есть глаза, уши, хвост. Рыжик, уйди пожалуйста! Извините, у любой мыши есть, по крайней мере, две кнопки. В новых моделях бывают и три, и пять кнопок. Также большинство мышей снабжается колесом прокрутки. Попробуйте отложить свою мышь в сторону и поработать с клавиатурой. Думаю, что 90% читателей будет чувствовать себя неуютно, не ощущая под руками округлые края кусочка пластика. И хотя Microsoft настоятельно рекомендует программистам дублировать операции мыши с помощью клавиш клавиатуры, тем не менее многие программы без мышки работать не в состоянии.

10.1. Рисование

С помощью мыши можно рисовать. Наверняка, вы не раз проделывали эту операцию в стандартном графическом редакторе Paint, который входит в состав любой Windows. Мы попробуем написать свой графический редактор. Сначала рассмотрим процесс рисования мышью. Рисование начинается с помещения указателя мыши на специально отведенную область для рисования. Это может быть клиентская область окна нашей программы или графическое поле. Следующий этап — нажатие кнопки мыши. Причем допускается нажатие как левой, так и правой кнопки мыши. В некоторых редакторах используются разные варианты, в зависимости от задумок разработчика. После нажатия кнопки мы перемещаем мышь в другое место, не отпуская кнопки. Во время перемещения за указателем остается след, позволяющий нам контролировать движения мыши. И, наконец, последняя операция — отпускание кнопки мыши. Естественно, рисование тут же прекращается. Описанный процесс можно повторять неограниченное число раз. Таким образом, рисование заключается в обработке событий мыши `MouseDown`, `MouseMove` и `MouseUp`. Вооружившись полученными знаниями,

приступим к созданию графического редактора. Для простого примера мы ограничимся одной формой и одной кнопкой. При помощи кнопки мы будем стирать не слишком удачные результаты наших экспериментов — проект `SimplePaint` (листинг 10.1).

Листинг 10.1. Рисование при помощи мыши

```
'--------------------------------------
' Рисуем мышью © 2004 Александр Климов
'--------------------------------------
    ' Координаты мыши в момент нажатия
    Dim x_md, y_md As Integer

    ' Будем рисовать пером синего цвета и толщиной 3 пиксела
    Dim p As New Pen(Color.Blue, 3)

    ' Отслеживаем нажатия кнопки, чтобы знать,
    ' когда пользователь заканчивает рисовать
    Dim bPaint As Boolean

    Private Sub Form1_MouseDown(ByVal sender As Object, ByVal e As
System.Windows.Forms.MouseEventArgs) Handles MyBase.MouseDown
        ' Начинаем рисовать
        bPaint = True

        ' Координаты мыши в момент нажатия
        x_md = e.X
        y_md = e.Y
    End Sub

    Private Sub Form1_MouseMove(ByVal sender As Object, ByVal e As
System.Windows.Forms.MouseEventArgs) Handles MyBase.MouseMove
        If bPaint Then
            ' Создадим объект Graphics для рисования
            Dim g As Graphics = CreateGraphics()

            ' Определяем координаты мыши при перемещении
            Dim x_mm As Integer = e.X
            Dim y_mm As Integer = e.Y
```

```
            ' Рисуем
            g.DrawLine(p, x_md, y_md, x_mm, y_mm)

            ' Передаем координаты мыши в глобальные переменные
            x_md = x_mm
            y_md = y_mm
            g.Dispose()
        End If
    End Sub

    Private Sub Form1_MouseUp(ByVal sender As Object, ByVal e As
System.Windows.Forms.MouseEventArgs) Handles MyBase.MouseUp
        bPaint = False
    End Sub

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles butClear.Click
        Dim g2 As Graphics = CreateGraphics()
        g2.Clear(MyBase.BackColor)
        g2.Dispose()
    End Sub
```

На рис. 10.1 вы видите изображение мышки, которую я нарисовал за несколько секунд. Вы говорите, что это не похоже на мышь? Да вы просто не разбираетесь в живописи!

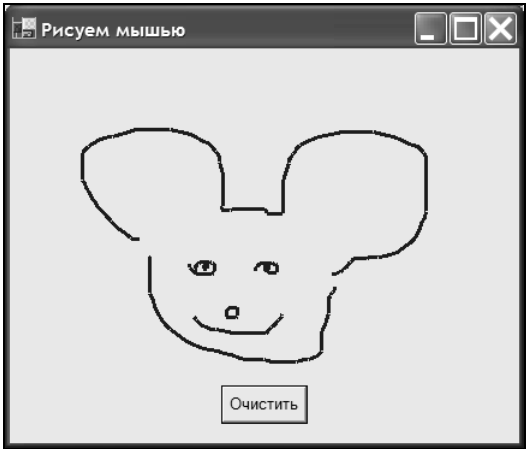

Рис. 10.1. Простейший редактор

Анекдот в тему

Приходит программист домой, а к нему кошка подбегает и начинает ласться, лизать руку, мурлыкать. Жена удивленно спрашивает:

– Что это с кошкой случилось? Почему она тебе руку лижет?

– Как чего? Рука то мышкой пахнет!

10.2. Прямые линии

Но любой графический редактор позволяет рисовать не только произвольные кривые, выводимые дрожащей рукой, но и прямые линии. Процесс создания прямых линий практически схож с принципом рисования, который мы только что рассмотрели. Но я внес небольшие изменения, чтобы показать другие способы. В частности, вместо отдельных переменных для координат мыши я применил структуру `Point` — проект `LinesGDIPlus` (листинг 10.2).

Листинг 10.2. Рисование прямых линий

```
'--------------------------------------
' Линии с помощью GDI+ © 2004 Александр Климов
'--------------------------------------
    Private g As Graphics

    Private DrawPen As Pen
    Private ErasePen As Pen

    Private ptsStart As Point
    Private ptsPrevious As Point
    Dim bPaint As Boolean

        InitializeComponent()

        'Добавьте свою инициализацию после вызова InitializeComponent()
        g = Me.CreateGraphics()

        ' Перо для стирания предыдущей линии
        ErasePen = New System.Drawing.Pen(Me.BackColor)
        ' Перо для рисования основной линии
        DrawPen = New System.Drawing.Pen(Color.Blue)
```

```
    Private Sub Form1_MouseDown(ByVal sender As Object, ByVal e As
System.Windows.Forms.MouseEventArgs) Handles MyBase.MouseDown
        bPaint = True

        ' Сохраняем начальные координаты мыши
        ptsStart.X = e.X
        ptsStart.Y = e.Y

        ' Сохраняем конечные координаты.
        ptsPrevious = ptsStart
    End Sub

    Private Sub Form1_MouseMove(ByVal sender As Object, ByVal e As
System.Windows.Forms.MouseEventArgs) Handles MyBase.MouseMove
        If bPaint Then
            ' Текущие координаты мыши.
            Dim ptsCurrent As Point

            ' Сохраняем текущие координаты конечной точки линии
            ptsCurrent.X = e.X
            ptsCurrent.Y = e.Y

            ' Рисуем линию, предварительно стирая предыдущие линии.
            g.DrawLine(ErasePen, _
                ptsStart.X, ptsStart.Y, ptsPrevious.X, ptsPrevious.Y)
            g.DrawLine(DrawPen, _
                ptsStart.X, ptsStart.Y, ptsCurrent.X, ptsCurrent.Y)

            ' Сохраняем текущую конечную точку линии для следующей линии.
            ptsPrevious = ptsCurrent
        End If
    End Sub

    Private Sub Form1_MouseUp(ByVal sender As Object, ByVal e As
System.Windows.Forms.MouseEventArgs) Handles MyBase.MouseUp
        bPaint = False
    End Sub
```

Вначале я объявил несколько глобальных переменных. Переменные `ErasePen` и `DrawPen` служат для рисования и стирания линий. Координаты

положения мыши теперь содержатся в структурах ptsStart и ptsPrevious. Инициализация графического объекта и перьев происходит в конструкторе New формы. Далее используем код из предыдущего примера и обрабатываем события MouseDown, MouseMove и MouseUp.

Но на самом деле полученный пример выглядит не слишком профессионально. Запустите проект и попробуйте сделать следующее. Начертите любую прямую линию. Она нарисуется без проблем. Теперь будьте внимательны. Щелкните мышью рядом с этой линией и, удерживая кнопку, переместите мышь в другое место. За указателем мыши потянется тонкая линия. Попрежнему не отпуская кнопки, перетащите мышь в другое какое-нибудь место. Попробуйте покрутить мышью вокруг начальной точки линии, стараясь зацепить первую нарисованную линию. Вы заметите, что при пересечении старой линии новой происходит частичное стирание этой линии. А происходит это потому, что GDI+ не поддерживает операцию XOR, которая позволяла бы восстанавливать стираемые точки линии. Решение данной проблемы лежит в применении графических функций Windows API, которое мы рассмотрим в следующем проекте.

10.3. Рисование линий с помощью функций Windows API

Итак, для проекта LinesAPI нам понадобятся несколько функций Windows API и структура Point, используемая некоторыми функциями. Задержимся на этой структуре. Во-первых, объявлять ее, во избежание ошибок, нужно перед объявлениями функций. Во-вторых, чтобы не было конфликтов с структурой .NET System.Drawing.Point, нужно изменить ее имя на другое, например, на POINTAPI (листинг 10.3).

Примечание

В Visual Basic 6.0 также приходилось использовать имя POINTAPI, так как там тоже присутствовало ключевое слово Point.

Листинг 10.3. Рисование линий с помощью Windows API

```
'--------------------------------------
' Линии с помощью функций WinAPI © 2004 Александр Климов
'--------------------------------------
    Private Structure POINTAPI
        Public x As Int32
        Public y As Int32
    End Structure
```

```
Private Declare Auto Function SetROP2 Lib "gdi32.dll" _
    Alias "SetROP2" (ByVal hDC As IntPtr, _
    ByVal nDrawMode As Int32) As Int32

Private Declare Auto Function MoveToEx Lib "gdi32.dll" _
    Alias "MoveToEx" (ByVal hDC As IntPtr, _
    ByVal x As Int32, ByVal y As Int32, _
    ByRef lpPoint As POINTAPI) As Boolean

Private Declare Auto Function LineTo Lib "gdi32.dll" _
    Alias "LineTo" (ByVal hDC As IntPtr, _
    ByVal x As Int32, ByVal y As Int32) As Boolean

Private Declare Auto Function CreatePen Lib "gdi32.dll" _
    Alias "CreatePen" (ByVal nPenStyle As Int32, _
    ByVal nWidth As Int32, ByVal crColor As Int32) As IntPtr

Private Declare Auto Function SelectObject Lib "gdi32.dll" _
    Alias "SelectObject" (ByVal hDC As IntPtr, _
    ByVal hObject As IntPtr) As IntPtr

Private Declare Function DeleteObject Lib "gdi32.dll" _
    Alias "DeleteObject" (ByVal hObject As IntPtr) As Boolean

Private Const R2_NOT As Int32 = 6
Private Const R2_XORPEN As Int32 = 7
Private Const PS_DOT As Int32 = 2
Private g As Graphics

Private ptsStart As POINTAPI
Private ptsPrevious As POINTAPI
Dim bPaint As Boolean

Public Sub New()
    MyBase.New()

    InitializeComponent()

    g = Me.CreateGraphics()
End Sub
```

```
   Private Sub Form1_MouseDown(ByVal sender As Object, ByVal e As Sys-
tem.Windows.Forms.MouseEventArgs) Handles MyBase.MouseDown
        bPaint = True

        ptsStart.x = e.X
        ptsStart.y = e.Y

        ptsPrevious = ptsStart
    End Sub

        Private Sub Form1_MouseUp(ByVal sender As Object, ByVal e As
System.Windows.Forms.MouseEventArgs) Handles MyBase.MouseUp
        bPaint = False
    End Sub

    Private Sub Form1_MouseMove(ByVal sender As Object, ByVal e As
System.Windows.Forms.MouseEventArgs) Handles MyBase.MouseMove
        If bPaint Then
            ' Дескриптор контекста устройства для рисования
            Dim hDC As IntPtr
            ' Предыдущий режим, возвращаемый функцией SetROP2
            Dim Mix As Int32
            ' Временные переменные
            Dim ptsCurrent As POINTAPI
            Dim ptsOld As POINTAPI
            ' Возвращаемые значения функций MoveToEx и LineTo
            Dim Success As Boolean

            ' Сохраняем текущие координаты последней точки линии
            ptsCurrent.x = e.X
            ptsCurrent.y = e.Y

            ' Получим дескриптор контекста устройства
            hDC = g.GetHdc()
            ' Установим новый графический режим
            ' инвертирования текущего цвета
            Mix = SetROP2(hDC, R2_NOT)

            ' Перемещаемся в начальную точку линии
            Success = MoveToEx(hDC, ptsStart.x, ptsStart.y, ptsOld)
```

```
            ' Очищаем предыдущую линию
            Success = LineTo(hDC, ptsPrevious.x, ptsPrevious.y)

            ' Снова перемещаемся в начальную точку линии
            Success = MoveToEx(hDC, ptsStart.x, ptsStart.y, ptsOld)

            ' Рисуем новую линию
            Success = LineTo(hDC, ptsCurrent.x, ptsCurrent.y)

            ' Освобождаем ресурсы
            g.ReleaseHdc(hDC)

            ' Сохраняем текущие координаты конечной точки линии
            ' для следующей операции стирания
            ptsPrevious = ptsCurrent
        End If

    End Sub
```

Основной момент, на который следует обратить внимание, — это использование функции `SetRop2` с константой `R2_NOT`. Преимущество использования данной функции состоит в том, что функция автоматически инвертирует цвета и нам не нужно думать о перьях для перерисовки пересекаемых линий. На сайте **http://msdn.microsoft.com** я нашел другой пример, позволяющий использовать различные стили перьев. Все, что нам нужно, — это закомментировать код для `MouseMove` и вставить новый код:

```
        If bPaint Then
            Dim hDC As IntPtr
            Dim Mix As Int32
            Dim ptsCurrent As POINTAPI
            Dim ptsOld As POINTAPI
            Dim Success As Boolean

            ' Дескрипторы для перьев
            Dim hPen As IntPtr
            Dim hOldPen As IntPtr

            ptsCurrent.x = e.X
            ptsCurrent.y = e.Y
```

```
        hDC = g.GetHdc()

        ' Создадим перо с точечным пунктиром в один пиксел
        hPen = CreatePen(PS_DOT, 1, &HFF0000)

        ' Свяжем hPen с текущим контекстом устройства
        hOldPen = SelectObject(hDC, hPen)

        ' Установим графический режим
        Mix = SetROP2(hDC, R2_XORPEN)

        Success = MoveToEx(hDC, ptsStart.x, ptsStart.y, ptsOld)

        Success = LineTo(hDC, ptsPrevious.x, ptsPrevious.y)

        Success = MoveToEx(hDC, ptsStart.x, ptsStart.y, ptsOld)

        Success = LineTo(hDC, ptsCurrent.x, ptsCurrent.y)

        ' Восстанавливаем предыдущее перо
        hPen = SelectObject(hDC, hOldPen)

        ' Удаляем созданное перо для создания линий
        Success = DeleteObject(hPen)

        ' Освобождаем ресурсы
        g.ReleaseHdc(hDC)

        ptsPrevious = ptsCurrent
    End If
```

Но, оказывается, и это еще не все. Существует третий способ рисования линий, который описан на том же сайте Microsoft. Рассказ будет не полным, если я не расскажу об этом способе.

10.4. Рисование линий с помощью метода *DrawReversibleLine*

В пространстве имен `System.Windows.Forms` содержится класс `ControlPaint`, позволяющий работать со стандартными элементами управления. В частно-

сти, в данном классе содержится метод `DrawReversibleLine`, рисующий линию. Только учтите одно обстоятельство. Данный метод рисует линию прямо на экране, не признавая границ формы. Таким образом, вам следует позаботиться об ограничениях для движения мыши. Но в нашем примере (листинг 10.4) мы этого делать не будем, так как получаемый эффект весьма интересен, и, может быть, вы найдете ему применение в своих проектах. Наш новый проект будет носить имя `LinesControlPaint`. На этот раз мы немного видоизменим проект и будем рисовать при помощи только правой кнопки мыши (надо же отдохнуть пальчику, которым вы постоянно нажимаете на ее левую кнопку).

Листинг 10.4. Третий способ рисования линий

```
'--------------------------------------
' Линии с помощью DrawReversibleLine © 2004 Александр Климов
'--------------------------------------
    Private ptsStart As Point
    Private ptsPrevious As Point

    Private Sub Form1_MouseDown(ByVal sender As Object, ByVal e As
System.Windows.Forms.MouseEventArgs) Handles MyBase.MouseDown
        ' Если нажата правая кнопка
        If e.Button = MouseButtons.Right Then
            ' Сохраняем координаты начальной точки линии.
            ptsStart.X = e.X
            ptsStart.Y = e.Y

            ' Сохраняем предыдущие координаты конечной точки.
            ptsPrevious = ptsStart
        End If
    End Sub

    Private Sub Form1_MouseMove(ByVal sender As Object, ByVal e As
System.Windows.Forms.MouseEventArgs) Handles MyBase.MouseMove
        ' Если нажата правая кнопка
        If e.Button = MouseButtons.Right Then
            ' Переменные для координат конечной точки линии
            Dim ptsCurrent As Point

            ' Сохраним координаты
            ptsCurrent.X = e.X
            ptsCurrent.Y = e.Y
```

```
            ' Рисуем линию.
            ControlPaint.DrawReversibleLine(PointToScreen(ptsStart), _
               PointToScreen(ptsPrevious), Color.Coral)
            ControlPaint.DrawReversibleLine(PointToScreen(ptsStart), _
               PointToScreen(ptsCurrent), Color.Coral)

            ' Сохраняем текущие координаты конечной точки
            ' для следующего вызова MouseMove
            ptsPrevious = ptsCurrent
        End If
    End Sub
```

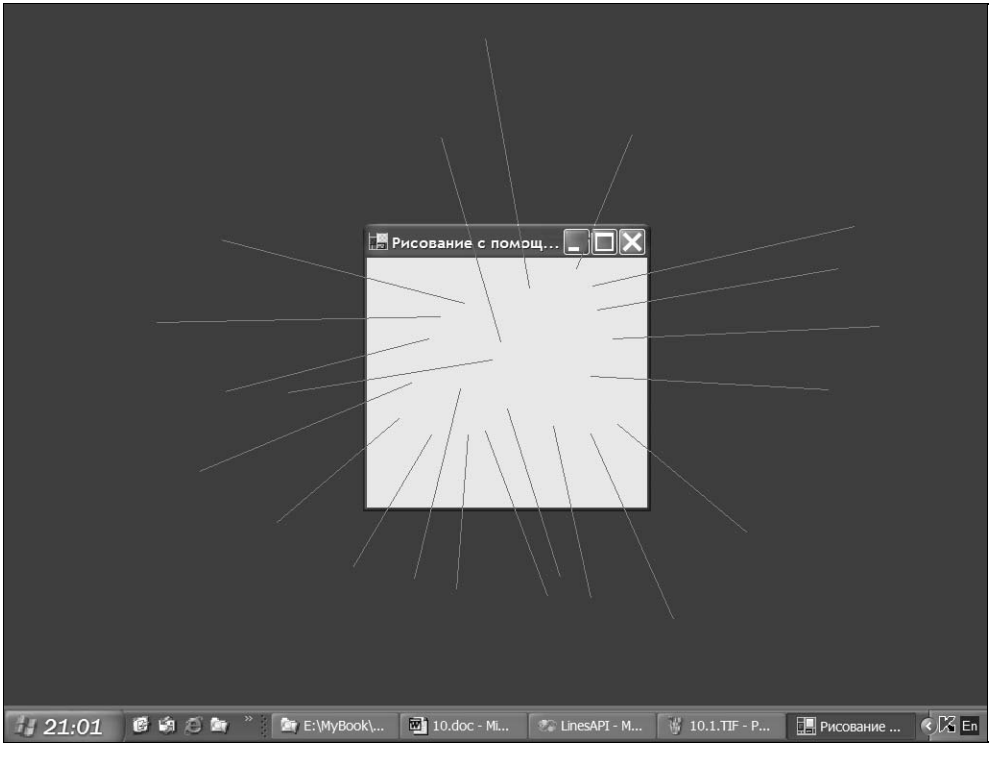

Рис. 10.2. Рисование линий за пределами формы

Комментарии в коде программы подробно объясняют процесс рисования линий. Ничего сложного тут нет. А теперь обещанный эффект. Запустите проект на выполнение и, удерживая правую кнопку мыши, тяните линию за пределы формы. К вашему изумлению, линия спокойно будет рисоваться и

там (рис. 10.2). Попробуйте сделать из вашей формы подобие ежика или пририсовать к ней усы. Итак, мы научились рисовать линии тремя способами. Лично мне больше нравится способ с применением функций Windows API. Какой способ подходит для решения ваших задач — решать вам. А я перехожу к следующему этапу — рисованию фигур.

10.5. Наложение рисунка на другой рисунок

В этом примере мы рассмотрим эффект наложения одного рисунка на другой. Причем форму рисунка будем определять сами с помощью движения мыши. Для нового проекта нам понадобятся четыре графических объекта `PictureBox`. Мы по-прежнему будем использовать события `MouseDown`, `MouseMove` и `MouseUp` для выбора области. Только на сей раз будем использовать события не формы, а элемента управления `PictureBox`. Наша задача — сохранить массив всех точек при движении мыши пользователем. С помощью сохраненного массива мы создадим траекторию, в которой будет содержаться часть выбранной картинки, и добавим эту копию в другую картинку в этих же координатах — проект `Overlay` (листинг 10.5).

Листинг 10.5. Наложение картинки

```
'--------------------------------------
' Наложение рисунка © 2004 Александр Климов
'--------------------------------------
Imports System.Drawing.Drawing2D
    ' Массив точек
    Private apoints() As Point
    Private m_MaxPoint As Integer

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        picFromClone.Image = DirectCast(picFrom.Image.Clone, Bitmap)
        picToClone.Image = DirectCast(picTo.Image.Clone, Bitmap)
    End Sub

    Private Sub picFrom_MouseDown(ByVal sender As Object, ByVal e As
System.Windows.Forms.MouseEventArgs) Handles picFrom.MouseDown

        ' Очищаем предыдущие выделения областей
        picFrom.Image = DirectCast(picFromClone.Image.Clone, Bitmap)
```

```
        ' Сохраняем начальную точку
        m_MaxPoint = 0
        ReDim apoints(m_MaxPoint)
        apoints(m_MaxPoint).X = e.X
        apoints(m_MaxPoint).Y = e.Y
    End Sub

    Private Sub picFrom_MouseMove(ByVal sender As Object, ByVal e As
System.Windows.Forms.MouseEventArgs) Handles picFrom.MouseMove
        ' Если ничего не выбрано, то ничего не делаем
        If apoints Is Nothing Then Exit Sub

        ' Сохраняем новую точку
        m_MaxPoint += 1
        ReDim Preserve apoints(m_MaxPoint)
        apoints(m_MaxPoint).X = e.X
        apoints(m_MaxPoint).Y = e.Y

        ' Рисуем линию из точек
        Dim g As Graphics = picFrom.CreateGraphics
        g.DrawLine(Pens.Yellow, _
            apoints(m_MaxPoint).X, _
            apoints(m_MaxPoint).Y, _
            apoints(m_MaxPoint - 1).X, _
            apoints(m_MaxPoint - 1).Y)
    End Sub

    Private Sub picFrom_MouseUp(ByVal sender As Object, ByVal e As
System.Windows.Forms.MouseEventArgs) Handles picFrom.MouseUp
        ' Если ничего не выбрано, то ничего не делаем
        If apoints Is Nothing Then Exit Sub

        ' Закрываем траекторию
        If (apoints(0).X <> apoints(m_MaxPoint).X) Or _
           (apoints(0).Y <> apoints(m_MaxPoint).Y) _
        Then
            ' Сохраняем новую точку
            m_MaxPoint += 1
```

```
            ReDim Preserve apoints(m_MaxPoint)
            apoints(m_MaxPoint).X = apoints(0).X
            apoints(m_MaxPoint).Y = apoints(0).Y
        End If

        ' Добавляем все точки в траекторию
        Dim area_path As New GraphicsPath(FillMode.Winding)
        area_path.AddLines(apoints)

        ' Устанавливаем границы выделения
        Dim selectbmp As Bitmap = DirectCast(picFromClone.Image.Clone,
        Bitmap)
        Dim g_select As Graphics = Graphics.FromImage(selectbmp)
        g_select.DrawPath(Pens.Orange, area_path)
        picFrom.Image = selectbmp

        ' Копируем выделенный рисунок во второй рисунок
        Dim resultbmp As Bitmap = DirectCast(picToClone.Image.Clone,
        Bitmap)
        Dim g_result As Graphics = Graphics.FromImage(resultbmp)

        ' Отсекаем траекторию
        g_result.SetClip(area_path)
        ' Копируем картинку
        g_result.DrawImage(picFromClone.Image, 0, 0)

        ' Выводим результат
        picTo.Image = resultbmp

        ' Обновляем данные
        apoints = Nothing

    End Sub
```

Как видно из рис. 10.3, блюдо с холодцом было скопировано с новогоднего стола в квартиру, где проживает кошка Кристина. Как говорится, от нашего стола к вашему.

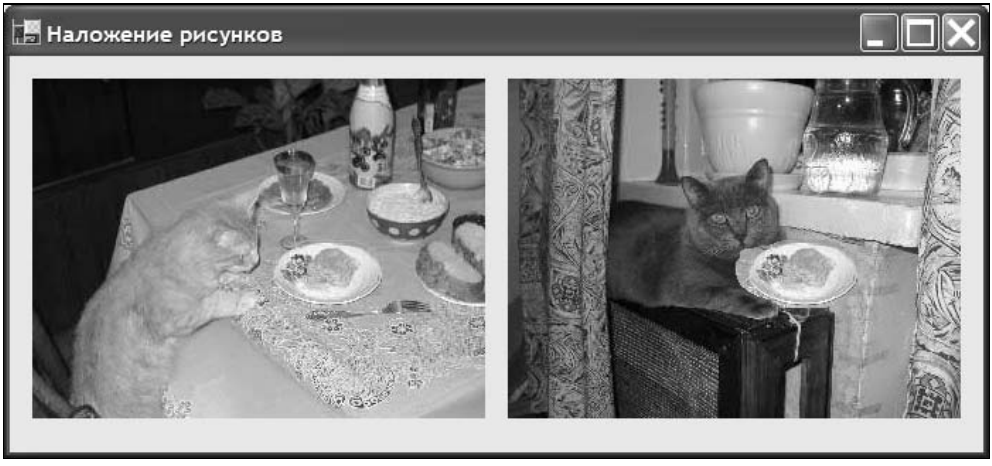

Рис. 10.3. Наложение одной картинки на другую

10.6. Паутинка

С помощью мыши можно рисовать не только примитивные линии и квадратики, но и самую настоящую паутину! Для нашего проекта `Web` понадобятся следующие приготовления. Сделаем фон формы черного цвета, так как паутина будет у нас белого цвета. Для указателя мыши присвоим свойству `Cursor` значение `Cross`. Также поместите на форму надпись `Label` и присвойте ему текст `Нажмите на кнопку мыши и, удерживая ее, подвигайте мышь`. Теперь можно переходить к кодированию. Сначала объявим две переменные на уровне класса формы (листинг 10.6).

Листинг 10.6. Создание паутины

```
'---------------------------------------
' Паутина © 2004 Александр Климов
'---------------------------------------
    Private pts As Point = Point.Empty

    ' Флаг, разрешающий рисовать паутинку
    Dim bDraw As Boolean

    Private Sub Form1_MouseDown(ByVal sender As Object, ByVal e As
System.Windows.Forms.MouseEventArgs) Handles MyBase.MouseDown
```

```
        bDraw = True
    End Sub

    Private Sub Form1_MouseMove(ByVal sender As Object, ByVal e As
System.Windows.Forms.MouseEventArgs) Handles MyBase.MouseMove
        If bDraw Then
            Dim g As Graphics = CreateGraphics()
            DrawWeb(g, BackColor, pts)
            pts = New Point(e.X, e.Y)
            DrawWeb(g, Color.White, pts)
            g.Dispose()
        End If
    End Sub

    Private Sub Form1_MouseUp(ByVal sender As Object, ByVal e As
System.Windows.Forms.MouseEventArgs) Handles MyBase.MouseUp
        bDraw = False
        Dim g As Graphics = CreateGraphics()
        g.Clear(BackColor)
        g.Dispose()
    End Sub

    Private Sub DrawWeb(ByVal g As Graphics, ByVal clr As Color, ByVal pt
As Point)
        Dim cx As Integer = ClientSize.Width
        Dim cy As Integer = ClientSize.Height
        Dim p As New Pen(clr)

        g.DrawLine(p, pt, New Point(0, 0))
        g.DrawLine(p, pt, New Point(cx \ 4, 0))
        g.DrawLine(p, pt, New Point(cx \ 2, 0))
        g.DrawLine(p, pt, New Point(3 * cx \ 4, 0))
        g.DrawLine(p, pt, New Point(cx, 0))
        g.DrawLine(p, pt, New Point(cx, cy \ 4))
        g.DrawLine(p, pt, New Point(cx, cy \ 2))
        g.DrawLine(p, pt, New Point(cx, 3 * cy \ 4))
        g.DrawLine(p, pt, New Point(cx, cy))
        g.DrawLine(p, pt, New Point(3 * cx \ 4, cy))
        g.DrawLine(p, pt, New Point(cx \ 2, cy))
```

```
        g.DrawLine(p, pt, New Point(cx \ 4, cy))
        g.DrawLine(p, pt, New Point(0, cy))
        g.DrawLine(p, pt, New Point(0, cy \ 4))
        g.DrawLine(p, pt, New Point(0, cy \ 2))
        g.DrawLine(p, pt, New Point(0, 3 * cy \ 4))
    End Sub
```

При перемещении мыши вызывается процедура `DrawWeb`, рисующая паутину. Обратите внимание, что процедура вызывается дважды. Первый раз для стирания предыдущей паутины, а второй вызов уже рисует саму паутину на экране. Смысл процедуры состоит в соединении линиями текущей позиции указателя мыши с краями формы. Попробуйте подвигать указатель мыши, не отпуская кнопки мыши. Вы увидите, что паутинка подвижна и следует за мышью (рис. 10.4).

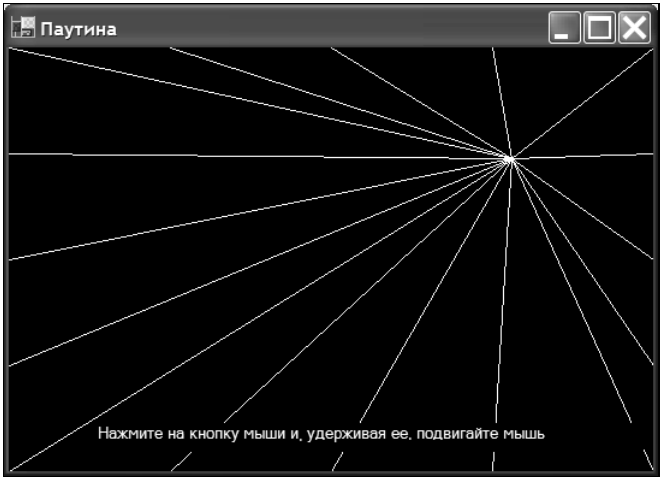

Рис. 10.4. Создание паутины

10.7. Перемещение указателя мыши

Для перемещения мыши используется свойство `Cursor.Position`, позволяющее устанавливать новую позицию для указателя мыши. Если мы хотим заставить двигаться мышь по замысловатой траектории, то можно воспользоваться свойством `PathData.Points`. Данное свойство позволяет получить координаты всех точек, входящих в траекторию. Таким образом, можно создать траекторию в виде окружности и заставить указатель крутиться вдоль

этой фигуры. Давайте посмотрим, как это реализовано в проекте `MovingMouse` (листинг 10.7).

Листинг 10.7. Перемещения указателя мыши

```
'---------------------------------------
' Перемещение мыши © 2004 Александр Климов
'---------------------------------------
Imports System.Drawing.Drawing2D

    Sub MouseFigure(ByVal times As Integer)
        ' Процедура, позволяющая перемещать курсор
        ' по заданной траектории
        ' Параметр times - количество повторений
        Dim gp As New GraphicsPath

        ' Размеры экрана
        Dim ScreenSize As Rectangle = Screen.GetBounds( _
                                          New Point(0, 0))

        ' Выберем произвольный прямоугольник в области экрана
        Dim rect As New Rectangle( _
                        CInt(ScreenSize.Width / 2) - 100, _
                        CInt(ScreenSize.Height / 2) - 100, 200, 200)

        ' Добавим в траекторию окружность
        gp.AddEllipse(rect)

        ' Разобьем окружность на серию линий
        gp.Flatten()

        ' Цикл повтороений движений мыши
        Dim repeat As Integer
        For repeat = 0 To times - 1

            Dim i As Integer
            ' Проходим через все точки траектории
            For i = 0 To gp.PathData.Points.Length - 1
```

```
            ' Организуем небольшую задержку
            System.Threading.Thread.Sleep(10)

            ' Перемещаем курсор
            Cursor.Position = New Point( _
                                  CInt(gp.PathData.Points(i).X), _
                                  CInt(gp.PathData.Points(i).Y))
        Next
    Next

    ' Освобождаем ресурсы
    gp.Dispose()
End Sub

Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
    MouseFigure(5)
End Sub
```

Запустите проект и нажмите на кнопку с текстом **Начать движение мыши**. Курсор мыши пять раз опишет окружность. Для усиления эффекта можете включить шлейф, тянущийся за мышью. Откройте панель управления и выберите апплет **Мышь**. Перейдите на вкладку **Параметры указателя** и установите флажок **Отображать след указателя мыши**. Данную технику перемещения указателя мыши можно использовать при создании демонстрационных приложений. Например, вы решили создать урок, рассказывающий об окнах, и вам надо обвести курсором границы окна на экране. Чуточку изменив предыдущий пример, вы легко выполните эту задачу (листинг 10.8).

Листинг 10.8. Границы формы

```
Sub AroundForm()
    ' Процедура, которая позволяет обвести границы формы мышью
    Dim gp As New GraphicsPath

    gp.AddLine(Me.DesktopLocation.X + 1, _
               Me.DesktopLocation.Y + 1, _
               Me.DesktopLocation.X + Me.Size.Width - 1, _
               Me.DesktopLocation.Y - 1)
```

```
        gp.AddLine(Me.DesktopLocation.X + Me.Size.Width - 1, _
                   Me.DesktopLocation.Y - 1, _
                   Me.DesktopLocation.X + Me.Size.Width - 1, _
                   Me.DesktopLocation.Y + Me.Size.Height - 1)

        gp.AddLine(Me.DesktopLocation.X + Me.Size.Width - 1, _
                   Me.DesktopLocation.Y + Me.Size.Height - 1, _
                   Me.DesktopLocation.X + 1, _
                   Me.DesktopLocation.Y + Me.Size.Height + 1)

        gp.AddLine(Me.DesktopLocation.X + 1, _
                   Me.DesktopLocation.Y + Me.Size.Height + 1, _
                   Me.DesktopLocation.X + 1, _
                   Me.DesktopLocation.Y + 1)

        Dim i As Integer
        ' Проходим через все точки траектории
        For i = 0 To gp.PathData.Points.Length - 1

            ' Организуем небольшую задержку
            System.Threading.Thread.Sleep(500)

            ' Перемещаем курсор
            Cursor.Position = New Point( _
                                  CInt(gp.PathData.Points(i).X), _
                                  CInt(gp.PathData.Points(i).Y))
        Next

        ' Освобождаем ресурсы
        gp.Dispose()
    End Sub

    Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button2.Click
        AroundForm()
    End Sub
```

10.8. Советы и рекомендации

- Работа с мышью настолько разнообразна, что вам придется постоянно возвращаться к этой теме. Например, вы можете написать графический редактор, используя примеры 10.1, 10.2, 10.3, дополнив его возможностью работать не только с графическими примитивами, но и с изображениями.
- В *главе 13* я покажу программу-шутку, в которой глаза кота будут следить за перемещениями мыши. Не пропустите!

ЧАСТЬ IV

ФОРМЫ
И ЭЛЕМЕНТЫ УПРАВЛЕНИЯ

Глава 11

Формы

Форма является фундаментом для разработки программ с графическим интерфейсом. Именно на форме мы располагаем все элементы управления, выводим тексты, рисуем фигуры. Обычно программисты сосредотачиваются на этих элементах. Но и сама форма имеет большой потенциал для решения многих задач. Поэтому, прежде чем перейти к нетрадиционному применению элементов управления, мы сначала остановимся на возможностях формы.

11.1. Форма-невидимка

Возможно, вам хорошо известно произведение знаменитого писателя Льюиса Кэрролла "Алиса в стране чудес". Там присутствовал такой персонаж, как Чеширский кот. Его главной особенностью было постепенное растворение в воздухе. Попробуем частично воспроизвести эту ситуацию (пока только на экране монитора). В VB .NET 2003 у форм появилось новое свойство `Opacity`, управляющее прозрачностью формы. Применение этого свойства с выдумкой может украсить вашу программу. Вот один из примеров использования свойства прозрачности формы (проект `TransparentForm`). Разместим на форме элемент управления `HScrollBar` (горизонтальный ползунок), который будет отвечать за прозрачность формы, а также кнопку `butExit` и таймер `Timer1`, которые будут закрывать программу с эффектом. Основной код сосредоточен в событии `HScrollBar1_Scroll` (листинг 11.1).

Листинг 11.1. Создание прозрачной формы

```
'---------------------------------------
' Прозрачная форма © 2004 Александр Климов
'---------------------------------------
    Private Sub HScrollBar1_Scroll(ByVal sender As System.Object, ByVal e
As System.Windows.Forms.ScrollEventArgs) Handles HScrollBar1.Scroll
        Me.Opacity = 0.1 + HScrollBar1.Value / 100
```

```
        Me.Text = Me.Opacity.ToString
    End Sub
```

Запустите проект и подвигайте ползунок вправо-влево. В заголовке формы вы увидите текущее состояние прозрачности формы, а сама форма может стать практически прозрачной (рис. 11.1). Я ввел в код ограничение на прозрачность, установив минимальное значение прозрачности `0.1`, чтобы не допустить полного исчезновения формы. Можно использовать это интересное свойство и при закрытии программы. В этом случае надо не закрывать сразу свое приложение, а дать программе возможность сначала сделать свое окно прозрачным (листинг 11.2).

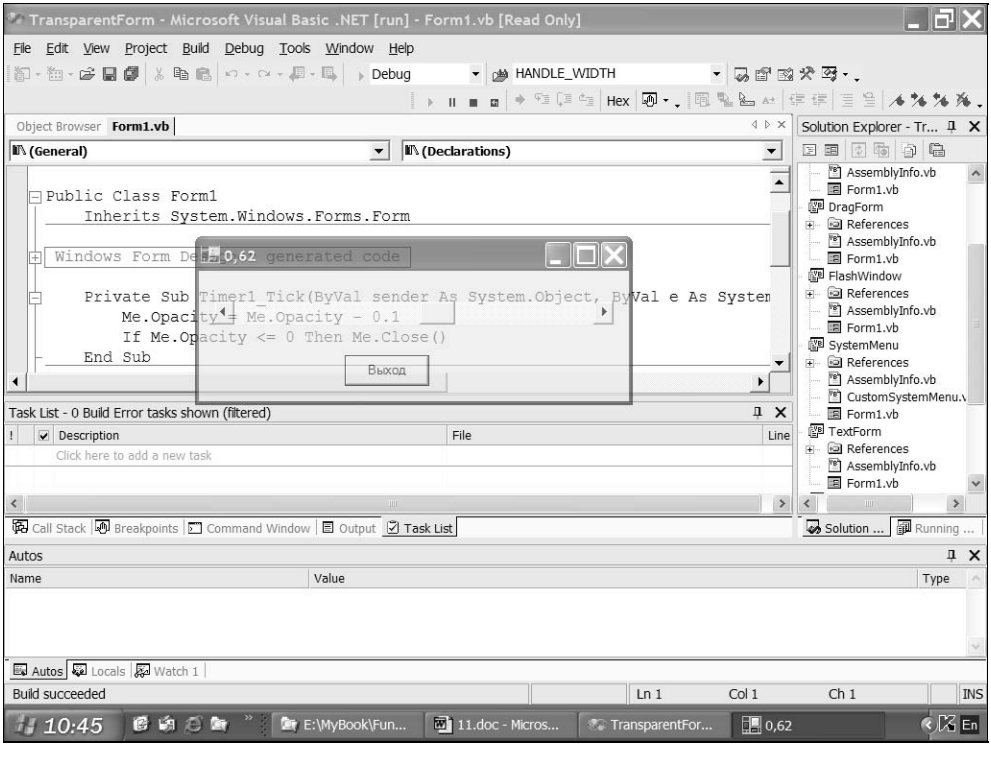

Рис. 11.1. Пример полупрозрачной формы

Листинг 11.2. Эффект при закрытии формы

```
    Private Sub Timer1_Tick(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Timer1.Tick
```

```
        Me.Opacity = Me.Opacity - 0.1
        If Me.Opacity <= 0 Then Me.Close()
    End Sub

    Private Sub butExit_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles butExit.Click
        Timer1.Enabled = True
    End Sub
```

11.2. Перемещение формы за заголовок

Рассмотрим еще один пример с прозрачностью формы. В некоторых случаях требуется обработать ситуацию нажатия кнопки мыши на заголовке формы. В этом случае окну посылается сообщение Windows `WM_NCLBUTTONDOWN` (при отпускании кнопки мыши используется сообщение `WM_NCLBUTTONUP`). Можно перехватывать эти сообщения и использовать в своих целях. Например, когда мы будем перетаскивать форму за ее заголовок, то она временно станет полупрозрачной. Код очень маленький, но эффективный — проект `TitleBarClick` (листинг 11.3).

Листинг 11.3. Перемещение формы за его заголовок

```
'-----------------------------------------
' Перемещение за заголовок формы © 2004 Александр Климов
'-----------------------------------------
    Private Const WM_NCLBUTTONDOWN As Long = &HA1
    Private Const WM_NCLBUTTONUP As Long = &HA0

    Protected Overrides Sub WndProc(ByRef m As
System.Windows.Forms.Message)
        ' Если нажата левая кнопка мыши на заголовке формы
        If CLng(m.Msg) = WM_NCLBUTTONDOWN Then
            ' Устанавливаем степень прозрачности формы
            If Me.Opacity <> 0.5 Then Me.Opacity = 0.5
            ' Кнопка отпущена
        ElseIf CLng(m.Msg) = WM_NCLBUTTONUP Then
            If Me.Opacity <> 1.0 Then Me.Opacity = 1.0
        End If
```

```
        MyBase.WndProc(m)
    End Sub

End Class
```

Запустите пример и попробуйте потаскать форму по всему экрану. Вы увидите, что при перемещении форма становится полупрозрачной.

11.3. Активная форма

Ну, если уж мы затронули тему использования полупрозрачности формы, то вот вам еще один удобный пример. Когда у вас на экране находятся несколько открытых форм, это может вызвать ощущение некоторого дискомфорта. Я предлагаю следующий способ. Можно отловить сообщение о том, что окно программы стало неактивным, и сделать его полупрозрачным. И наоборот, когда окно вновь станет активным, вернуть ему первоначальное состояние — проект `ActiveForm` (листинг 11.4).

Листинг 11.4. Проверка состояния активности формы

```
'--------------------------------------
' Активная форма © 2004 Александр Климов
'--------------------------------------
    Protected Overrides Sub WndProc(ByRef m As
System.Windows.Forms.Message)
        MyBase.WndProc(m)

        Const WM_ACTIVATEAPP As Integer = &H1C
        If m.Msg = WM_ACTIVATEAPP Then
            If m.WParam.ToInt32() <> 0 Then
                ' Окно стало активным
                If Me.Opacity <> 1.0 Then Me.Opacity = 1.0
            Else
                ' Окно стало неактивным
                ' Устанавливаем степень прозрачности формы
                If Me.Opacity <> 0.5 Then Me.Opacity = 0.5
            End If
        End If
    End Sub
```

Запустите программу вместе с несколькими другими программами. Переключитесь на другую программу. Вы увидите, что ваше приложение, потеряв фокус, стало полупрозрачным (рис. 11.2). На мой взгляд, весьма интересное применение полупрозрачности, которое можно предложить Биллу Гейтсу для использования в будущих версиях Windows!

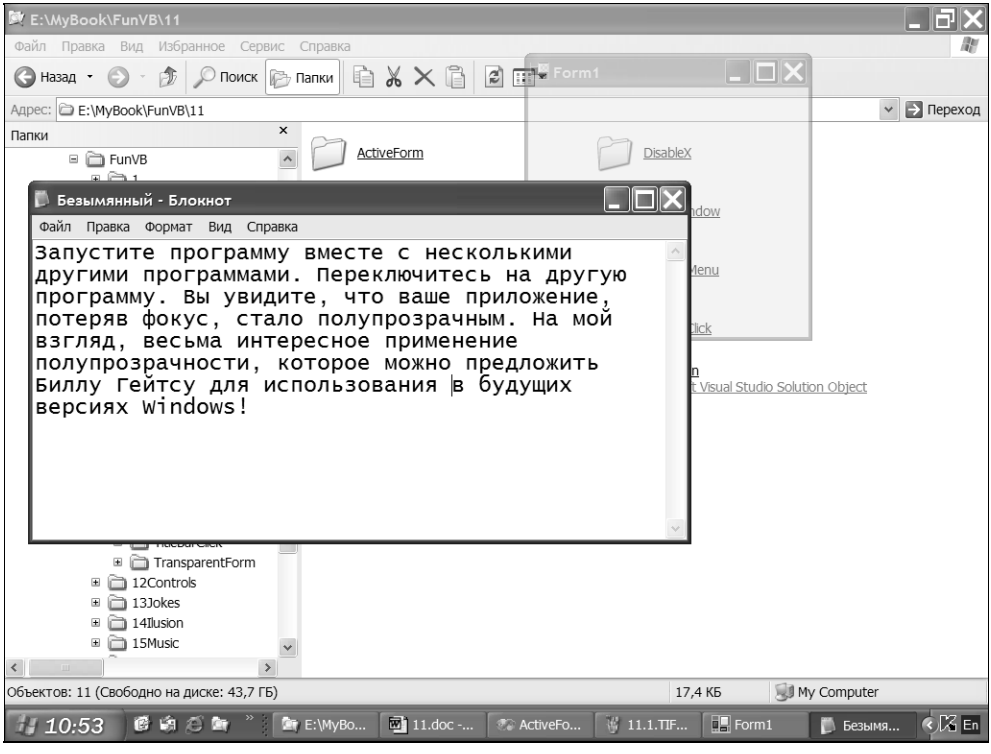

Рис. 11.2. Полупрозрачное неактивное окно

11.4. Форма в виде текста

Мы все привыкли, что форма имеет вид прямоугольника (В Windows XP появился новый вид окна с закругленными верхними углами). Но если стандартный вид вас не устраивает, то вы можете без проблем создать окно произвольной формы. Например, в виде текста. Давайте попробуем! Пример реализован в проекте `TextForm` (листинг 11.5). В основе примера лежит использование очень мощного и интересного класса `GraphPath` и метода `Region`.

Листинг 11.5. Создание формы в виде текста

```
'--------------------------------------
' Окно в виде текста © 2004 Александр Климов
'--------------------------------------
Imports System.Drawing.Drawing2D

    Private Sub Form1_Paint(ByVal sender As Object, ByVal e As
System.Windows.Forms.PaintEventArgs) Handles MyBase.Paint
        MyBase.BackColor = Color.Cyan
        Dim gp As New GraphicsPath
        Dim m_Text As String = "Котенок"
        gp.AddString(m_Text, New FontFamily("Tahoma"), _
                 FontStyle.Bold, 70, New Point(35, 5), _
                 StringFormat.GenericDefault)
        MyBase.Region = New Region(gp)
    End Sub

    Private Sub Form1_DoubleClick(ByVal sender As Object, ByVal e As
System.EventArgs) Handles MyBase.DoubleClick
        MyBase.Close()
    End Sub
```

Сначала я присваиваю форме цвет `Cyan` и создаю экземпляр класса `GraphicsPath`. В качестве текста, форму которого примет наше окно, я выбрал слово "Котёнок". Если вы заядлый собачник, то можете использовать слово "Щенок". В методе `AddString` я устанавливаю все необходимые свойства для шрифта — название, стиль, координаты вывода и формат. Обратите внимание на второй параметр структуры `Point`, я чуть позже вернусь к нему. Теперь достаточно присвоить свойству `Region` созданную траекторию, и все лишнее будет отброшено. Останется только заданный текст.

Вернемся к параметру структуры `Point`. Если вы не присваивали свойству `FormBorderStyle` формы значение `None`, то без труда заметите часть заголовка формы в первой букве слова. Можно увидеть, что две точки буквы "ё" тоже попали в область заголовка. Попадание отдельных участков в заголовок формы облегчает нам решение некоторых задач. В частности, вы можете ухватиться за этот оставшийся заголовок и перетащить форму в другое место. Можно вызвать системное меню и выбрать команду **Закрыть**, чтобы выйти из программы (рис. 11.3). Безусловно, вы можете создать форму без заголовка или выбрать область региона чуть пониже, чтобы заголовок не попал на участок текста. Но в этом случае вам необходимо предусмотреть

различные ситуации. Например, чтобы выход из программы осуществлялся при двойном щелчке мыши. Как перетаскивать форму за любое место, будет рассказано далее в этой главе. Используя вместо метода `AddString` схожие методы с префиксом `Add`, можно создавать формы не только в виде текста, но и в виде различных фигур.

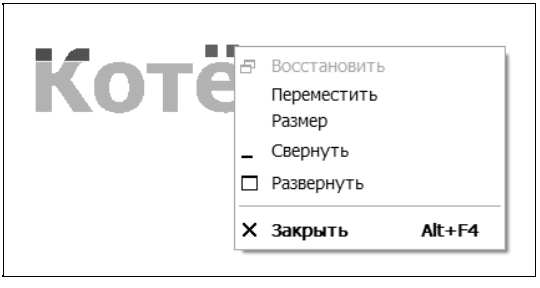

Рис. 11.3. Создание формы в виде текста

11.5. Буксировка формы не за ее заголовок

Как вы знаете, окно программы можно перемещать по экрану, уцепившись за его заголовок. Но иногда возникает необходимость буксировки окна нажатием мыши в любом другом месте окна. Например, у окна может и не быть заголовка. Здесь нам на помощь придут функции Windows API. Для проекта `DragForm` нам понадобятся надпись `Label` и кнопка `Button`. Присвойте элементам следующие свойства:

- установки для формы: `FormBorderStyle = None`;
- установки для надписи `Label`:
  - `Name = lblDescription`;
  - `BackColor = LightBlue`;
  - `Text = Нажмите левую кнопку мыши и, удерживая ее, перетащите форму в другое место`;
- установки для кнопки `Button`:
  - `Name = butExit`;
  - `Text = Выход`.

Надпись информирует пользователя о том, что ему нужно сделать для просмотра эффекта. Так как буксировка в нашем примере будет работать только

при нажатии на самом окне (нажатия на элементы управления не обрабатываются), то надпись выделим другим цветом для контраста. У нашей формы нет заголовка, а, следовательно, крестика закрытия программы в правом верхнем углу. Для того чтобы мы могли закрыть окно программы, я добавил на форму кнопку **Выход**, которая выполняет эту работу (листинг 11.6).

Листинг 11.6. Буксировка формы не за ее заголовок

```
'--------------------------------------
' Буксировка формы © 2004 Александр Климов
'--------------------------------------
  'Функции API и константы
  Private Declare Function ReleaseCapture Lib "user32" () As Long

  Private Declare Function SendMessage Lib "user32" Alias "SendMessageA _
          (ByVal hwnd As IntPtr _
           ByVal wMsg As Integer, _
           ByVal wParam As Integer, _
           ByVal lParam As Integer) As Integer

    Private Const WM_NCLBUTTONDOWN As Integer = &HA1
    Private Const HTCAPTION As Integer = 2

    Private Sub Form1_MouseDown(ByVal sender As Object _
           ByVal e As System.Windows.Forms.MouseEventArgs _
           Handles MyBase.MouseDown

        ' Только при нажатии левой кнопки мыши.
        If e.Button = MouseButtons.Left Then
            ReleaseCapture()
            SendMessage(Me.Handle, WM_NCLBUTTONDOWN, HTCAPTION, 0)
        End If
    End Sub
```

Так как обычно перетаскивание формы осуществляется левой кнопкой мыши, то в событии `MouseDown` проверяется условие:

```
If e.Button = MouseButtons.Left
```

И при положительном ответе осуществляется вызов функций Windows API, которые позволяют буксировать форму не только за ее заголовок, но и за

любое место в клиентской области. Запустите проект и проверьте способность окна перетаскиваться при нажатии мыши в любом месте формы.

11.6. Системное меню

У любой формы с заголовком есть системное меню. Все мы привыкли к стандартному его виду, а ведь это меню можно изменить. Для этого нам придется заняться вызовом функций Windows API. Запустим новый проект `SystemMenu`. Так как мы будем переопределять некоторые свойства формы, то создадим новый класс `CustomSystemMenu`, в котором разместим следующий код (листинг 11.7).

Листинг 11.7. Создание нового класса для изменения системного меню

```
'---------------------------------------
' Класс для создания системного меню © 2004 Александр Климов
'---------------------------------------
Public Class CustomSystemMenu
    Inherits System.Windows.Forms.NativeWindow
    Implements IDisposable

    ' Функции Windows API
    Private Declare Function GetSystemMenu Lib "user32" _
                             (ByVal hwnd As Int32, _
                              ByVal bRevert As Boolean) As Int32
    Private Declare Function AppendMenu Lib "user32" _
            Alias "AppendMenuA" (ByVal hMenu As Int32, _
                              ByVal wFlags As Int32, _
                              ByVal wIDNewItem As Int32, _
                              ByVal lpNewItem As String) As Int32

    ' Константы
    Private Const MF_STRING As Int32 = &H0       ' Строка меню
    Private Const MF_SEPARATOR As Int32 = &H800  ' Разделитель

    ' Сообщение Windows
    Private Const WM_SYSCOMMAND As Int32 = &H112

    ' Наш новый идентификатор для системного меню
    Private Const ID_ABOUT As Int32 = 1000
```

```
    ' Дескриптор системного меню
    Private hSystemMenu As Int32

    ' Дескриптор окна
    Private hParentHandle As Int32

    ' Текст для создаваемого пункта меню
    Private strAboutText As String = String.Empty

    Public Event ShowMsgBox()

    ' Конструктор для создания нового пункта меню
    ' Параметры:
    ' intWinHandle : Родительское окно для обработки сообщений
    ' и добавления нового пункта меню
    Public Sub New(ByVal intWinHandle As Int32, _
                ByVal strMenuItemText As String)

        Me.AssignHandle(New IntPtr(intWinHandle))

        hParentHandle = intWinHandle
        strAboutText = strMenuItemText

        ' Получим дескриптор системного меню
        hSystemMenu = GetSystemMenu(hParentHandle, 0)

        If CheckNewItem() = False Then
            Throw New Exception(_
          "Ошибка при попытке создания нового пункта системного меню")
        End If

    End Sub

    ' Обработка сообщения WM_SYSCOMMAND
    Protected Overrides Sub WndProc(ByRef m As
System.Windows.Forms.Message)

        Select Case m.Msg
            Case WM_SYSCOMMAND
                MyBase.WndProc(m)
```

```
            If m.WParam.ToInt32 = ID_ABOUT Then
                If hSystemMenu <> 0 Then
                    RaiseEvent ShowMsgBox()
                End If
            End If

        Case Else
            MyBase.WndProc(m)
    End Select

End Sub

' Освобождение ресурсов
Public Sub Dispose() Implements System.IDisposable.Dispose

    If Not Me.Handle.Equals(IntPtr.Zero) Then
        Me.ReleaseHandle()
    End If

End Sub

' Проверка на добавляемый новый пункт в системное меню
Private Function CheckNewItem() As Boolean
    Try
        Return AppendToSystemMenu(hSystemMenu, strAboutText)
    Catch ex As Exception
        Return False
    End Try
End Function

' Добавляем  разделитель и новый пункт в системное меню
' Параметры:
' intHandle : Дескриптор системного меню
' strText : Добавляемый текст
Private Function AppendToSystemMenu(ByVal intHandle As Int32, _
                            ByVal strText As String) As Boolean

    Try
        ' Добавляем разделитель
```

```
            Dim intRet As Int32 = AppendMenu _
                      intHandle, MF_SEPARATOR, 0, String.Empty)

            ' Теперь добавляем свою строчку
            intRet = AppendMenu(intHandle, MF_STRING, ID_ABOUT, strText)

            If intRet = 1 Then
                Return True
            Else
                Return False
            End If

        Catch ex As Exception
            Return False
        End Try
    End Function

End Class
```

В данном классе мы с помощью функций Windows API `GetSystemMenu` и `AppendMenu` добавляем разделитель и новый пункт в системное меню, а также отлавливаем сообщение Windows `WM_SYSCOMMAND`, которое посылается окну при использовании системного меню. Теперь переключаемся на главное окно программы `Form1`, где объявляем переменную `mySystemMenu` на уровне класса, которую можно инициализировать в процедуре `New` следующим образом:

```
mySystemMenu = New CustomSystemMenu(Me.Handle.ToInt32, "&О программе...")
```

Нам осталось только обработать событие `ShowMsgBox`, которое возникнет при выборе созданного пункта меню (листинг 11.8).

Листинг 11.8. Работа с новым пунктом системного меню

```
    ' Объявляем переменную
    Private WithEvents mySystemMenu As CustomSystemMenu

    Public Sub New()
        MyBase.New()

        'This call is required by the Windows Form Designer.
        InitializeComponent()
```

```
        'Add any initialization after the InitializeComponent() call
        mySystemMenu = New CustomSystemMenu(Me.Handle.ToInt32, "&О
программе...")
    End Sub

    Private Sub mySystemMenu_ShowMsgBox() _
                            Handles mySystemMenu.ShowMsgBox
        ' Обработка щелчка на созданном пункте меню
        MsgBox( _
          "Занимательное программирование" & vbNewLine & "А.Климов")
    End Sub
```

Запустите проект, щелкните мышью на значке приложения в верхнем левом углу и убедитесь в наличии нового пункта меню (рис. 11.4). Выберите созданный нами новый пункт меню и щелкните на нем мышью — у вас должно появиться стандартное окно с сообщением. Конечно, в своей практике вы не ограничитесь таким простым сообщением, а создадите полноценное диалоговое окно, в котором сообщите о своей гениальности. Не забудьте только в вашем окне мелким шрифтом поблагодарить автора этой книги за предоставленный пример. Знаете ли, доброе слово и кошке приятно.

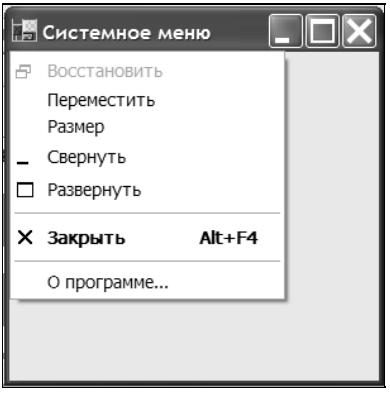

Рис.11.4. Новый пункт в системном меню

11.7. Мигание заголовка формы

Наверное, вы встречались с такой ситуацией в своей практике, когда вдруг начинал мигать заголовок окна, не имеющего фокуса. Тем самым окно привлекает ваше внимание, призывая сделать его активным. Подобный эффект

реализуется вызовом функций `FlashWindow` или `FlashWindowEx`. Я остановлюсь на более мощной и современной функции `FlashWindowEx`. Пример реализован в проекте `FlashWindow`. Для примера достаточно будет расположить на форме одну кнопку. Назовем кнопку `butFlash` и присвоим ей текст `Начать мигание`. Переключаемся в редактор кода. Для начала надо подключить пространство имен `System.Runtime.InteropServices` (листинг 11.9).

Листинг 11.9. Использование функции `FlashWindowEx`

```
'--------------------------------------
' Мигание окна © 2004 Александр Климов
'--------------------------------------
Imports System.Runtime.InteropServices

    ' Константы для структуры
    Private Const FLASHW_STOP As Integer = 0
    Private Const FLASHW_CAPTION As Integer = &H1
    Private Const FLASHW_TRAY As Integer = &H2
    Private Const FLASHW_ALL As Integer = (FLASHW_CAPTION Or FLASHW_TRAY)
    Private Const FLASHW_TIMER As Integer = &H4
    Private Const FLASHW_TIMERNOFG As Integer = &HC
    Private FLASHW_FLAGS As Integer

    ' Структура для функции
    Private Structure FLASHWINFO
        Dim cbSize As Integer
        Dim hwnd As IntPtr
        Dim dwFlags As Integer
        Dim uCount As Integer
        Dim dwTimeout As Integer
    End Structure

    Private Declare Function FlashWindowEx Lib "user32" _
                (ByRef pflashwininfo _
                As FLASHWINFO) As Integer

    Private Sub butFlah_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles butFlah.Click
        Dim intRetval
        Dim fwi As FLASHWINFO
```

```
        With fwi
            .cbSize = Marshal.SizeOf(fwi)
            .hwnd = Me.Handle
            .dwFlags = FLASHW_TRAY
            .dwTimeout = 0
            .uCount = 5
        End With

        intRetval = FlashWindowEx(fwi)
    End Sub
```

Функция `FlashWindowEx` опирается при своем вызове на структуру `FLASHINFO`. Данная структура использует различные константы. Я решил привести все возможные константы, хотя в нашем примере реально применяется только одна константа `FLASHW_TRAY`. При использовании этой константы выполняется вариант мигания заголовка окна только на панели задач. При нажатии на кнопку форма мигнет пять раз (переменная `uCount = 5`). Запустите проект на исполнение и нажмите на кнопку. Ваше окно должно замигать внизу на панели задач.

11.8. Убрать кнопку из заголовка формы

Весьма распространенный вопрос на форумах: "Как сделать недоступной кнопку на заголовке формы, а также убрать команду **Закрыть** из системного меню?" Для решения подобной задачи нам снова понадобится вызывать функции Windows API, работающие с системным меню — `GetSystemMenu`, `GetMenuItemCount`, `DrawMenuBar`, `RemoveMenu`, — проект `DisableX` (листинг 11.10).

Листинг 11.10. Скрытие значка из заголовка формы

```
'--------------------------------------
' Убрать кнопку X из заголовка формы © 2004 Александр Климов
'--------------------------------------
    ' константы
    Const MF_BYPOSITION = &H400&
    Const MF_REMOVE = &H1000&
```

```
    ' функции WinAPI
    Private Declare Function GetSystemMenu Lib "user32" (_
          ByVal hwnd As Integer, ByVal bRevert As Integer) As Integer

    Private Declare Function GetMenuItemCount Lib "user32" _
          (ByVal hMenu As Integer) As Integer

    Private Declare Function DrawMenuBar Lib "user32" _
          (ByVal hwnd As Integer) As Integer

    Private Declare Function RemoveMenu Lib "user32" _
          (ByVal hMenu As Integer, ByVal nPosition As Integer, _
           ByVal wFlags As Integer) As Integer

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        Dim hSysMenu As Integer
        Dim nCnt As Short

        ' Получим дескриптор системного меню формы
        ' (Восстановить, Переместить, Развернуть, Свернуть, Закрыть)
        hSysMenu = GetSystemMenu(Handle.ToInt32, False)

        If hSysMenu Then
            ' Считаем число пунктов в системном меню
            nCnt = GetMenuItemCount(hSysMenu)
                        If nCnt Then
                ' Отсчет от 0
                ' Пункт Закрыть является последним пунктом
                RemoveMenu(hSysMenu _
                           nCnt - 1, MF_BYPOSITION Or MF_REMOVE)
                ' Удаляем разделитель
                RemoveMenu(hSysMenu _
                          nCnt - 2, MF_BYPOSITION Or MF_REMOVE)

                ' Переририсовываем меню
                DrawMenuBar(Handle.ToInt32)
            End If
        End If
    End Sub
```

```
   Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
        Application.Exit()
    End Sub
```

Как правило, команда **Закрыть** является последней командой системного меню. Этой команде обычно предшествует разделитель. Таким образом, нам нужно просто подсчитать число всех команд меню и удалить две последние команды. Если вы запустите проект прямо сейчас, то увидите, что кнопка  заблокирована. Но у нас возникнет одна проблема. Мы не сможем теперь закрыть форму и выйти из программы. Чтобы исправить эту ситуацию, пришлось добавить на форму кнопку `Button1` и написать для кнопки код закрытия программы:

```
Application.Exit()
```

11.9. Хранители экрана

Хранители экрана (или более официальное название — *экранные заставки*) были созданы в то далекое время, когда мониторы были еще несовершенны. Долгое воспроизведение статической картинки выводило мониторы из строя. Подобная ситуация случалась, если пользователь отходил от компьютера на какое-то время. И тогда была предложена идея самопроизвольного включения хранителя экрана, если пользователь в течение определенного промежутка времени не пользовался мышью или клавиатурой. На данный момент это уже не актуально, поэтому все чаще мы и говорим: "Экранная заставка" вместо устаревшего "Хранитель экрана". Сейчас хранители экрана служат в основном для развлечения или для скрытия результатов своей работы от любопытных взглядов сотрудников (в этом случае хранитель экрана еще дополняется паролем).

Создавать хранители экрана на Visual Basic .NET очень просто. По своей сути это обычная программа с формой, но при этом обладающая некоторыми особенностями. Во-первых, хранитель экрана имеет расширение scr вместо стандартного exe. Во-вторых, он должен блокировать запуск других экземпляров. В-третьих, во время выполнения программы указатель мыши не должен отображаться на экране. В-четвертых, программа должна автоматически закрываться при нажатии любой клавиши или любом передвижении мыши (не говоря уж о нажатии на кнопку мыши). Кроме того, желательно, чтобы стандартный хранитель экрана обеспечивал предварительный просмотр и имел окно настроек, доступных через апплет панели управления **Экран**. Это основные принципы любого хранителя экрана. Как видите, принципы не слишком сложные.

Теоретически любая программа может работать как хранитель экрана. Но, тем не менее, стоит придерживаться некоторых правил. Форма хранителя экрана должна занимать весь экран, на форме не должно быть заголовка, на экран должно выводиться динамическое изображение. В дополнение к вышесказанному нужно добавить, что хранители экрана могут иметь дополнительные ключи командной строки:

- `/p` — хранитель экрана выводится в специальном окне просмотра. При нажатии на кнопку **Просмотр** должен запуститься сам хранитель экрана;
- `/c` — при нажатии на кнопку **Параметры** выводится специальное диалоговое окно настроек хранителя экрана;
- `/s` — хранитель экрана запускается в нормальном обычном режиме.

Теперь, когда мы познакомились с основными правилами, можно приступать к созданию собственных хранителей экрана.

11.9.1. Создание простого хранителя экрана

А начнем мы с создания самого простого хранителя. За основу я взял пример, который нашел на сайте Microsoft. В нем отражены принципы создания заставки в минимальном объеме, что вполне достаточно для первого знакомства. Во втором примере я предложу более серьезную разработку. Запустим проект. Установим сразу необходимые свойства для формы:

- `BackColor` = `Black`;
- `ControlBox` = `False`;
- `FormBorderStyle` = `None`;
- `MaximizeBox` = `False`;
- `MinimizeBox` = `False`;
- `ShowInTaskBar` = `False`;
- `WindowState` = `Maximized`.

Создавать свой хранитель экрана мы будем постепенно (проект `SimpleSsaver`). Периодически я буду советовать запускать проект на выполнение, чтобы проверить программу на наличие ошибок и лучше понять суть происходящего. Итак, вы выставили все необходимые свойства для формы. Теперь нам надо объявить несколько глобальных переменных, которые нам понадобятся при выполнении различных операций (листинг 11.11).

Листинг 11.11. Первые шаги в создании хранителя экрана

```
'--------------------------------------
' Хранитель экрана © 2004 Александр Климов
'--------------------------------------
```

```
' Объявляем глобальные переменные
' Объект Graphics, на котором будем рисовать
Private g As Graphics

' Объект Random для вывода случайных фигур
Private m_Random As New Random

' Для первого использования события MouseMove
Private m_IsActive As Boolean = False

' Для определения перемещения мыши
Private m_MouseLocation As Point

' Объект Options, содержащий информацию о выборе пользователя
Private m_Options As New Options
```

Как вы помните, хранитель экрана должен автоматически закрываться при движении мыши и при нажатиях на ее кнопки. Значит, надо писать код для событий `MouseMove` и `MouseDown` (листинг 11.12).

Листинг 11.12. Обработка событий мыши

```
    Private Sub Form1_MouseMove(ByVal sender As Object, ByVal e As Sys-
tem.Windows.Forms.MouseEventArgs) Handles MyBase.MouseMove
        '  При движении мыши хранитель экрана должен закрыться
        '  Но событие MouseMove возникает при самых простых движениях
        '  Таким образом, мы отслеживаем события,
        '  когда мышь действительно переместилась
        '  на несколько пикселов перед закрытием.
        '  Если MouseLocation имеет координаты 0,0,
        '  то перемещаем в настоящую позицию
        '  и сохраняем для дальнейшего использования.
        '  Иначе следим, переместил ли пользователь
        '  мышь по крайней мере на 10 пикселов.
        If Not m_IsActive Then
            Me.m_MouseLocation = New Point(e.X, e.Y)
            m_IsActive = True
        Else
            If Math.Abs(e.X - Me.m_MouseLocation.X) > 10 Or _
```

```
                Math.Abs(e.Y - Me.m_MouseLocation.Y) > 10 Then
                ' Если пользователь переместил мышь, то выходим
                Application.Exit()
            End If
        End If
    End Sub

    Private Sub Form1_MouseDown(ByVal sender As Object, ByVal e As
System.Windows.Forms.MouseEventArgs) Handles MyBase.MouseDown
        ' Выходим из программы при нажатии кнопки
        Application.Exit()
    End Sub
```

Для самопроверки снова запустим проект, убедимся, что все пока работает должным образом, и пойдем дальше. Наверное, почти любой хранитель экрана использует таймер, чтобы периодически менять изображения на экране. Размещаем на форме таймер, включаем его в событии `Form_Load`. Заодно создаем объект Graphics, который нам скоро пригодится (листинг 11.13).

Листинг 11.13. Подключение таймера

```
    Private Sub Form1_Load(ByVal sender As Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        ' Создаем объект Graphics для рисования.
        g = CreateGraphics()

        ' Включаем таймер
        Timer1.Enabled = True
    End Sub

    Private Sub Timer1_Tick(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Timer1.Tick
        DrawFigures()
    End Sub
```

Вся работа таймера сводится к выводу фигур на экране. Для этого мы создали отдельную процедуру `DrawFigures()` (листинг 11.14).

Листинг 11.14. Процедура для рисования фигур

```
    Private Sub DrawFigures()
        ' Процедура рисования фигур различных цветов и размеров
```

```
        ' Получим размеры экрана
        Dim cx As Integer = Me.Width
        Dim cy As Integer = Me.Height

        ' Координаты случайных точек
        Dim x1, x2, y1, y2 As Integer

        ' Прямоугольники для рисования фигур
        Dim myRect As Rectangle

        ' Цвет для фигур
        Dim myColor As Color

        ' Генерируем случайные числа для координат точек
        x1 = m_Random.Next(0, cx)
        x2 = m_Random.Next(0, cx)

        y1 = m_Random.Next(0, cy)
        y2 = m_Random.Next(0, cy)

        ' Создаем прямоугольник на основе случайных координат
        myRect = New Rectangle(Math.Min(x1, x2), Math.Min(y1, y2), _
                Math.Abs(x1 - x2), Math.Abs(y1 - y2))

        ' Получим цвет, выбранный случайным образом.
        myColor = Color.FromArgb(255, m_Random.Next(255), _
            m_Random.Next(255), m_Random.Next(255))

        ' Рисуем эллипс.
        g.FillEllipse(New SolidBrush(myColor), myRect)
    End Sub
```

В дальнейшем мы еще добавим новые строчки в эту процедуру, а пока можно снова запускать программу на выполнение. Теперь вы увидите вполне работоспособную заставку. На экране в случайном порядке рисуются эллипсы на черном фоне.

А теперь забудьте все, что я вам тут рассказывал. На самом деле запуск программы начинается не с события `Form_Load`, а с процедуры `Main`. Это

очень важная процедура (листинг 11.15). Именно в ней происходит обработка параметров командной строки, о которых говорилось ранее.

Листинг 11.15. Обработка параметров командной строки

```
<STAThread()> Shared Sub Main(ByVal args As String())
    '  С этой процедуры начинается выполнение программы
    '   Здесь происходит обработка параметров командной строки
    '   и настроек пользователя через кнопку Параметры
    '   в апплете Экран -> вкладка Заставка.
    '   Если параметры не используются,
    '   то запускается заставка в обычном режиме
    If args.Length > 0 Then
        '   Если параметры используются, то обрабатываем их.
        '   Windows распознает следующие ключи:
        '   "/s", "/p" или "/c"

        ' Настройки для предварительного просмотра.
        If args(0).ToLower = "/p" Then
            ' Пока оставим в покое

            ' Просто выходим из программы
            Application.Exit()
        End If

        ' Если у программы есть окно настроек.
        If args(0).ToLower.Trim().Substring(0, 2) = "/c" Then
            ' Создаем форму frmOptions и выводим на экран.
            Dim userOptionsForm As New frmOptions
            userOptionsForm.ShowDialog()

            ' Выходим
            Application.Exit()
        End If

        ' Если заставка просто должна запуститься
        If args(0).ToLower = "/s" Then
```

```
            ' Создаем основную форму и выводим его.
            Dim screenSaverForm As New Form1
            screenSaverForm.ShowDialog()

            ' Выходим при закрытии формы
            Application.Exit()
        End If

    Else
        '  Запускаем заставку. Эта ситуация используется
        '  когда пользователь запускает программу двойным щелчком
        '  Параметры командной строки не обрабатываются

        ' Создаем форму frmSceenSaver и выводим на экран.
        Dim screenSaverForm As New Form1
        screenSaverForm.ShowDialog()

        ' Выходим при закрытии формы
        Application.Exit()
    End If
End Sub
```

Я снабдил код процедуры подробными комментариями. Единственное, на что нужно обратить внимание, — в этой процедуре мы обращаемся к новой форме `frmOptions`, в которой пользователь может настраивать различные свойства заставки: вид и цвет фигур, скорость появления на экране. Поэтому добавляем в проект новую форму и располагаем на ней несколько элементов управления. Внешний вид окна настроек показан на экране (рис. 11.5). Для нас представляет интерес код для кнопки **OK** и события `Load` формы настроек. Именно там происходит сохранение пользовательских настроек на диск и считывание их из файла (листинг 11.16).

Листинг 11.16. Сохранение и извлечение настроек

```
    Private Sub btnOK_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnOK.Click
        ' Здесь происходит изменение настроек
        ' и сохранение их на диске.

        Dim myOptions As New Options
```

```
        If Me.optEllipses.Checked Then
            myOptions.Shape = "Ellipses"
        Else
            myOptions.Shape = "Rectangles"
        End If

        myOptions.IsTransparent = Me.chkTransparent.Checked
        myOptions.Speed = Me.cboSpeed.Text

        ' Сохраняем настройки на диск.
        myOptions.SaveOptions()

        ' Закрываем объект.
        Me.Close()

    End Sub


    Private Sub frmOptions_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        ' Загружаем текущие настройки и устанавливаем значения
        ' различных элементов управления.

        Dim myOptions As New Options
        myOptions.LoadOptions()

        Me.cboSpeed.Text = myOptions.Speed
        Me.chkTransparent.Checked = myOptions.IsTransparent

        If myOptions.Shape = "Ellipses" Then
            Me.optEllipses.Checked = True
        Else
            Me.optRectangles.Checked = True
        End If

    End Sub
```

Как вы могли догадаться по этим строчкам кода, наша заставка может рисовать не только эллипсы, но и прямоугольники, а также настраивать про-

зрачный цвет и скорость вывода на экран. Поэтому вносим небольшие изменения в процедуру `DrawFigures`:

```
    Private Sub DrawFigures()


        ' Если пользователь хочет использовать прозрачность,
        ' то дадим такую возможность.
        If m_Options.IsTransparent Then
            myColor = Color.FromArgb(m_Random.Next(255),
m_Random.Next(255), _
                m_Random.Next(255), m_Random.Next(255))
        Else
            ' Если прозрачный цвет не нужен,
            ' то альфа-составляющая равна 255.
            myColor = Color.FromArgb(255, m_Random.Next(255), _
                m_Random.Next(255), m_Random.Next(255))
        End If

        ' Рисуем эллипс.
        If m_Options.Shape = "Эллипсы" Then
            g.FillEllipse(New SolidBrush(myColor), myRect)
        Else
            ' Рисуем прямоугольник.
            g.FillRectangle(New SolidBrush(myColor), myRect)
        End If
```

Также необходимо добавить глобальную переменную, содержащую объект для настроек пользователя:

```
    ' Объект Options, содержащий информацию о выборе пользователя
    Private m_Options As New Options
```

Теперь необходимо внести изменения в событие `Form1_Load`:

```
   Private Sub Form1_Load(ByVal sender As Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        ...
        ' Загружаем сохраненные настройки.
        ' Обратите внимание, что если файл настроек не существует
        ' то он будет создан
        m_Options.LoadOptions()
```

```
        ' Установка скорости.
        Select Case m_Options.Speed
            Case "Медленная"
                Timer1.Interval = 500
            Case "Быстрая"
                Timer1.Interval = 100
            Case Else
                Timer1.Interval = 200
        End Select
        …
    End Sub
```

Все настройки хранителя экрана можно сохранять в любом удобном вам виде — в обычном текстовом документе или в реестре. Microsoft, например, предложила в своем примере заставки (который послужил основой для моей программы) использовать возможности языка XML. Надо сказать, что корпорация активно продвигает язык XML в массы. Возможно, вам стоит присмотреться к данному языку для дальнейшего использования в своих проектах. А пока мы перейдем к завершающей стадии нашего примера. Для создания файла настроек с использованием XML нам понадобится добавить в проект новый класс. Назовем его `Options` (файл Options.vb). Я включил все необходимые комментарии в код класса, поэтому расписывать пример не буду. Думаю, разберетесь сами (листинг 11.17).

Листинг 11.17. Файл настроек хранителя экрана

```
'********************
' файл Options.vb
' Настройки для хранителя экрана
'********************
Option Strict On

Imports System.Xml.Serialization
Imports System.Xml
Imports System.IO

' Данный класс содержит информацию о настройках

<Serializable()> Public Class Options
```

```
' Устанавливаем переменные и значения по умолчанию
Private m_Speed As String = "Быстрая"
Private m_Shape As String = "Эллипсы"
Private m_IsTransparent As Boolean = True

' Установим местоположение файла настроек
Private OptionsPath As String =  _
      Environment.CurrentDirectory & "\ScreenSaver_Options.opt"

' --- Свойства класса ---

' Использует ли заставка прозрачный цвет
' при выводе фигур.
Public Property IsTransparent() As Boolean
    Get
        Return m_IsTransparent
    End Get
    Set(ByVal Value As Boolean)
        m_IsTransparent = Value
    End Set
End Property

' Какие фигуры рисуем на экране
Public Property Shape() As String
    Get
        Return m_Shape
    End Get

    Set(ByVal Value As String)
        m_Shape = Value
    End Set
End Property

' Скорость вывода фигур на экране
Public Property Speed() As String
    Get
        Return m_Speed
    End Get
```

```
    Set(ByVal Value As String)
        m_Speed = Value
    End Set
End Property

' --- Методы класса ---

' Если файл настроек существует,
' то функция возвращает 'true', 'false' - в другом случае.
Public Function IsOptionFileExisting() As Boolean
    Dim myIO As New System.IO.FileInfo(OptionsPath)
    Return myIO.Exists()
End Function

' Функция загружает пользовательские настройки.
' Сначала идет проверка существования файла настроек
' Если файл существует, то настройки загружаются из него.
' Если файл не существует, то он создается со значениями по умолчанию
Public Sub LoadOptions()

    Dim myOptions As New Options   ' Объект Options

    ' Проверка существования файла или создание нового
    If myOptions.IsOptionFileExisting() Then

        ' Загружаем настройки
        ' Создаем XmlSerializer для получения значений настроек
        Dim mySerializer As New XmlSerializer(GetType(Options))

        ' Создаем StreamReader для указания файла настроек
        Dim myTextReader As New StreamReader(OptionsPath)

        ' Создаем XmlTextReader для чтения настроек.
        Dim myXmlReader As New Xml.XmlTextReader(myTextReader)

        ' Сначала проверяем, что файл работает в нужном формате.
        If mySerializer.CanDeserialize(myXmlReader) Then
```

```
                myOptions = CType(mySerializer.Deserialize(myXmlReader),
                Options)
            Else
                ' Сохраняем новый файл настроек
                myOptions.SaveOptions()
            End If

            ' Закрываем объект IO
            myXmlReader.Close()
            myTextReader.Close()

            ' Устанавливаем свойства для различных настроек из файла
            ' (или используем по умолчанию, если файл не существует)
            Me.Speed = myOptions.Speed
            Me.IsTransparent = myOptions.IsTransparent
            Me.Shape = myOptions.Shape

        End If
    End Sub

    ' Процедура сохранения новых настроек
    Public Sub SaveOptions()

        Dim myWriter As New System.IO.StreamWriter(OptionsPath)

        Dim myXmlSerializer As New XmlSerializer(Me.GetType())

        myXmlSerializer.Serialize(myWriter, Me)

        myWriter.Close()

    End Sub

End Class
```

Вот и готова наша первая заставка. Осталось сделать несколько важных шагов по установке заставки в систему. Сначала создайте исполняемый файл (меню **Build-Build ScreenSaver**). Найдите этот файл (в подпапке bin) и поменяйте расширение файла на scr. Щелкните правой кнопкой на полу-

чившемся файле и выберите команду **Install** (можно также вручную скопировать файл в папку Windows\system32). Теперь откройте апплет **Экран** (**Пуск | Панель управления | Экран**) и перейдите на вкладку **Заставка**. Найдите в выпадающем списке название вашей заставки и нажмите кнопку **Параметры**. У вас должно появиться диалоговое окно настроек. Установите требуемые значения и закройте окно. Теперь можете щелкнуть на кнопке **Просмотр** и посмотреть программу в действии.

11.9.2. Вторая заставка

Я думаю, что одного примера не достаточно для изучения всех особенностей создания хранителей экрана. Тем более, что Microsoft не до конца использовала возможности, предоставляемые стандартной заставкой. В частности, заставка не обрабатывает один из параметров командной строки, позволяющий видеть заставку в маленьком окошке на вкладке **Заставка** апплета **Экран**. Кроме того, мы не написали процедуру закрытия заставки при нажатии любой клавиши на клавиатуре. Поэтому я приведу еще один пример, частично позаимствованный с сайта Рода Стивенса **http://www.vb-helper.com/** (проект `RodStephens_Ssaver`). Так как принципы создания заставок однотипны, я не буду снова подробно описывать все строчки кода, а остановлюсь на некоторых из них (также почитайте комментарии в коде). Кстати, заставка рисует мячи, которые отскакивают от стенок экрана (об отражениях будет рассказано в других главах).

Первое, что бросается в глаза, это вынос главной запускающей процедуры `Main` в отдельный модуль SubMain.vb. Там же находятся объявления функций Windows API и констант, необходимых для нашего примера (листинг 11.18).

Листинг 11.18. Летающие мячи

```
'--------------------------------------
' Хранитель экрана © 2004 Александр Климов
' Оригинальная версия проекта находится на сайте Рода Стивенса
' http://www.vb-helper.com
'--------------------------------------
Module SubMain
    Public Const SWP_NOACTIVATE = &H10
    Public Const SWP_NOZORDER = &H4
    Public Const SWP_SHOWWINDOW = &H40
    Public Const GWL_STYLE = -16
    Public Const WS_CHILD = &H40000000
```

```
    Public Const GWL_HWNDPARENT = -8
    Public Const HWND_TOP = 0

    Public Structure RECT
        Public left As Integer
        Public top As Integer
        Public right As Integer
        Public bottom As Integer
    End Structure

    Public Declare Function GetClientRect Lib "user32" ( _
        ByVal hwnd As Integer, _
        ByRef lpRect As RECT) As Integer

    Public Declare Function GetWindowLong Lib "user32" Alias
"GetWindowLongA" ( _
        ByVal hwnd As Integer, _
        ByVal nIndex As Integer) As Integer

    Public Declare Function SetWindowLong Lib "user32" Alias
"SetWindowLongA" ( _
        ByVal hwnd As Integer, _
        ByVal nIndex As Integer, _
        ByVal dwNewInteger As Integer) As Integer

    Public Declare Function SetWindowPos Lib "user32" ( _
        ByVal hwnd As Integer, _
        ByVal hWndInsertAfter As Integer, _
        ByVal x As Integer, _
        ByVal y As Integer, _
        ByVal cx As Integer, _
        ByVal cy As Integer, _
        ByVal wFlags As Integer) As Integer

    Public Declare Function SetParent Lib "user32" ( _
        ByVal hWndChild As Integer, _
        ByVal hWndNewParent As Integer) As Integer

    Public Enum ActionType
        actConfigure
```

```
    actPreview
    actRun
End Enum

Public m_Action As ActionType
Public Sub Main(ByVal args As String())
    ' обработка параметров командной строки
    If args.Length = 0 Then
        m_Action = ActionType.actRun
    Else
        Select Case args(0).ToLower().Substring(0, 2)
            Case "/p"
                m_Action = ActionType.actPreview
            Case "/c"
                m_Action = ActionType.actConfigure
            Case "/s"
                m_Action = ActionType.actRun
            Case Else
                m_Action = ActionType.actRun
        End Select
    End If


    Select Case m_Action
        Case ActionType.actRun
            ' Обычный режим
            Dim canvas As New frmCanvas
            Application.Run(canvas)
        Case ActionType.actConfigure
            ' Окно настроек
            Dim dlg_config As New frmConfig
            Application.Run(dlg_config)
        Case ActionType.actPreview
            ' Окно предварительного просмотра
            Dim canvas As New frmCanvas
            SetForm(canvas, args(1))
            Application.Run(canvas)
    End Select
End Sub
```

```
    ' Меняем родителя формы для окна предварительного просмотра
    Private Sub SetForm(ByRef frm As Form, ByRef arg As String)
        Dim style As Integer
        Dim preview_hwnd As Integer = Integer.Parse(CType(arg, String))
        Dim r As New RECT

        ' Получим размеры окна предварительного просмотра.
        GetClientRect(preview_hwnd, r)

        With frm
            .WindowState = FormWindowState.Normal
            .FormBorderStyle = FormBorderStyle.None
            .Width = r.right
            .Height = r.bottom
        End With

        ' Добавляем стиль WS_CHILD.
        style = GetWindowLong(frm.Handle.ToInt32, GWL_STYLE)
        style = style Or WS_CHILD
        SetWindowLong(frm.Handle.ToInt32, GWL_STYLE, style)

        ' Меняем родителя для окна.
        SetParent(frm.Handle.ToInt32, preview_hwnd)

        ' Устанавливаем значение GWL_PARENT к окну предварительного
          просмотра.
        SetWindowLong(frm.Handle.ToInt32, GWL_HWNDPARENT, preview_hwnd)

        ' Позиция формы в окне предварительного просмотра.
        SetWindowPos(frm.Handle.ToInt32, 0, r.left, 0, r.right, r.bottom, _
            SWP_NOACTIVATE Or SWP_NOZORDER Or SWP_SHOWWINDOW)
    End Sub
End Module
```

Вам нужно внимательно разобрать процедуру `SetForm`. Именно там идет формирование клона заставки, который будет отображаться в апплете **Экран**. Форма `frmConfig` представляет собой окно настроек заставки (листинг 11.19). В ней содержится текстовое поле, в котором пользователь сможет

самостоятельно устанавливать число мячей, которые будут летать по экрану (пользователю ужасно нравится самому настраивать программы).

Листинг 11.19. Код для формы `frmConfig`

```
Public Class frmConfig
    Inherits System.Windows.Forms.Form

#Region " Windows Form Designer generated code "
...
#End Region

    ' Выводим текущие настройки
    Private Sub Config_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        txtNumBalls.Text = GetSetting("Bouncer", "Settings", "NumBalls",
        "20")
    End Sub

    ' Сохраняем новые настройки.
    Private Sub cmdOk_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles cmdOk.Click
        Try
            SaveSetting("Bouncer", "Settings", "NumBalls",
            CLng(txtNumBalls.Text))
        Catch exc As Exception
        End Try

        Close()
    End Sub

    Private Sub cmdCancel_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles cmdCancel.Click
        Close()
    End Sub
End Class
```

Число мячей, выбранных неугомонным пользователем, записывается в реестр при нажатии на кнопку **OK**. При нажатии на кнопку **Отмена** мы просто закрываем окно. Основной код программы содержится в форме `frmCanvas`

(листинг 11.20). Так как ничего принципиально нового там не приведено, я воздержусь от комментариев.

Листинг 11.20. Код для формы `frmCanvas`

```
Public Class frmCanvas
    Inherits System.Windows.Forms.Form

    Private m_Xmax As Integer
    Private m_Ymax As Integer

    ' (X, Y) - координаты верхнего левого угла мячей.
    ' D - диаметр мячей.
    Private m_NumBalls As Long
    Private m_Color() As Color
    Private m_D() As Single
    Private m_X() As Single
    Private m_Y() As Single
    Private m_Vx() As Single
    Private m_Vy() As Single

    Private m_BufferBitmap As Bitmap
    Private m_BufferGraphics As Graphics
    Private m_Graphics As Graphics
    Private m_Random As New Random

    ' Прячем курсор и включаем таймер.
    Private Sub frmCanvas_Load(ByVal sender As Object, ByVal e As
System.EventArgs) Handles MyBase.Load

        If m_Action = ActionType.actRun Then
            Cursor.Hide()
        Else
            RemoveHandler Me.KeyDown, AddressOf frmCanvas_KeyDown
            RemoveHandler Me.MouseMove, AddressOf frmCanvas_MouseMove
            RemoveHandler Me.MouseDown, AddressOf frmCanvas_MouseDown
        End If
```

```
    ' Сохраняем размеры экрана.
    m_Xmax = Width
    m_Ymax = Height

    ' Получим число мячей.
    m_NumBalls = CLng(GetSetting("Bouncer", "Settings", "NumBalls",
    "20"))

    ' Создаем мячи.
    ReDim m_Color(m_NumBalls)
    ReDim m_D(m_NumBalls)
    ReDim m_X(m_NumBalls)
    ReDim m_Y(m_NumBalls)
    ReDim m_Vx(m_NumBalls)
    ReDim m_Vy(m_NumBalls)

    Const MAX_VEL As Single = 0.025
    Const MIN_D_FACTOR As Single = 0.025
    Const MAX_D_FACTOR As Single = 0.1
    For i As Integer = 0 To m_NumBalls - 1
        m_Color(i) = Color.FromArgb(&HFF000000 Or
        QBColor(m_Random.Next(1, 15)))
        m_D(i) = RandomSingle(m_Ymax * MIN_D_FACTOR, m_Ymax *
        MAX_D_FACTOR)
        m_X(i) = RandomSingle(1, m_Xmax - m_D(i))
        m_Y(i) = RandomSingle(1, m_Ymax - m_D(i))
        m_Vx(i) = RandomSingle(-m_Ymax * MAX_VEL, m_Ymax * MAX_VEL)
        m_Vy(i) = RandomSingle(-m_Ymax * MAX_VEL, m_Ymax * MAX_VEL)
    Next i

    m_BufferBitmap = New Bitmap(m_Xmax, m_Ymax)
    m_BufferGraphics = Graphics.FromImage(m_BufferBitmap)

    ' Создаем объект Graphics для формы.
    m_Graphics = CreateGraphics()

    tmrMove.Enabled = True
End Sub
```

```
    ' Возвращает случайное число
    Private Function RandomSingle(ByVal min_value As Single, ByVal
max_value As Single) As Single
        Return min_value + (max_value - min_value) *
m_Random.NextDouble()
    End Function
```

Изучайте код самостоятельно и обязательно впоследствии напишите свою заставку, которая удивит весь мир. Удачи вам! Также я рекомендую вам посещать мой сайт, где обязательно буду выкладывать интересные эффектные заставки.

11.10. Советы и рекомендации

Форма — достаточно консервативный элемент графического интерфейса. Поэтому здесь трудно предложить вам интересные рекомендации. Другое дело — создание хранителей экрана. Здесь вы можете дать волю своей фантазии. Посмотрите, какие экранные заставки находятся на вашем компьютере, и попробуйте создать аналогичные программы. Не обязательно добиваться полного сходства, ваша задача — поломать голову над решением интересных задач.

Глава 12

Элементы управления

Элементами управления называют объекты, которые мы располагаем на форме (кнопки, надписи, текстовые поля) или в специальном контейнере (таймер). Среди программистов также принято называть их *контролами* (от английского *control*). Вряд ли вы найдете программу, не использующую хотя бы один элемент управления. С выходом каждой новой версией Visual Basic число встроенных элементов управления постоянно увеличивается. Кроме того, программист может сам создать новый элемент управления для решения собственных задач, либо приобрести его у сторонних разработчиков. В этой главе мы рассмотрим несколько нетривиальных способов использования элементов управления в программах.

12.1. Элемент управления произвольной формы

А начнем мы с абстрактного элемента. Стандартный вид элементов управления иногда утомляет меня. Хочется чего-то необычного. Например, почему бы не сделать кнопку овальной или треугольной? Здесь нам на помощь придет свойство `Region`, которое отбрасывает ненужные области элемента управления. В свое время я создал еще для VB 6.0 ActiveX-элемент `OvalButton`, который еще можно найти на моем сайте. Для создания подобного элемента пришлось перелопатить тонны документации и использовать функции Windows API. Интерес к полученному продукту был огромен. Существовали и аналогичные коммерческие варианты. Но теперь все это в прошлом. Создание элемента управления произвольной формы стало удивительно легкой задачей.

Продемонстрируем сказанное на примере проекта `RoundControls`. Поместим на форму кнопку под именем `butRound`. В результате наших действий она должна стать овальной. Пусть это произойдет при нажатии на саму кнопку (листинг 12.1).

Листинг 12.1. Создание овальной кнопки

```
'--------------------------------------
' Создание овальной кнопки © 2004 Александр Климов
'--------------------------------------
Imports System.Drawing.Drawing2D

    Private Sub butRound_Click(ByVal sender As System.Object, _
                ByVal e As System.EventArgs) Handles butRound.Click

        Dim gp As New GraphicsPath
        Dim g As Graphics = CreateGraphics()

        ' Создадим новый прямоугольник с размерами кнопки
        Dim newRectangle As Rectangle = butRound.ClientRectangle

        ' Уменьшим размеры прямоугольника
        newRectangle.Inflate(-3, -3)

        'Создадим эллипс с полученными размерами
        gp.AddEllipse(newRectangle)
        butRound.Region = New Region(gp)

        ' Нарисуем окантовку для круглой кнопки
        g.DrawEllipse(New Pen(Color.Gray, 2), butRound.Left + 1, _
                    butRound.Top + 1, _
                    butRound.Width - 3, butRound.Height - 3)

        ' Освобождаем ресурсы
        g.Dispose()
    End Sub
```

Небольшое пояснение. Уменьшение прямоугольника понадобилось для того, чтобы не видеть окантовки кнопки. В этом случае кнопка выглядит более аккуратно. Вот так просто можно создать элемент управления произвольной формы. Аналогично можно поступать и с другими элементами управления.

12.2. Нестандартный вид элементов управления

Иногда хочется отойти от стандартного вида элементов управления, придав им некоторую индивидуальность. Вот небольшой пример легкой модификации стандартного элемента `Checkbox` в проекте `RedCheckBox` (листинг 12.2). Поместим на форму обычный `CheckBox` и напишем следующий код.

Листинг 12.2. Нестандартный `CheckBox`

```
'--------------------------------------
' Нестандартный CheckBox © 2004 Александр Климов
'--------------------------------------
    Private Sub Form1_Paint(ByVal sender As Object, ByVal e As
System.Windows.Forms.PaintEventArgs) Handles MyBase.Paint
        Dim g As Graphics = e.Graphics
        Dim p As New Pen(Color.Red, 3)
        Dim ctrl As Control

        For Each ctrl In Me.Controls
            If TypeOf ctrl Is CheckBox Then
                g.DrawRectangle(p, New _
                  Rectangle(ctrl.Location, ctrl.Size))
            End If
        Next
    End Sub
```

Запустив проект, вы увидите, что элемент `CheckBox` обведен красной рамочкой. Естественно, подобный прием также можно применить практически к любому элементу управления.

12.3. Текстовое поле *TextBox*

Текстовое поле — один из самых используемых элементов управления в приложениях. Область применения — текстовые редакторы, поля для ввода паролей, информационные сообщения, калькуляторы и многое другое.

12.3.1. Автоматическая прокрутка в конец текста

Если в многострочном текстовом поле нужно добавить слово в конец текста, то возникает необходимость программно прокрутить текст до конца, чтобы увидеть это вставленное слово. Этого легко добиться тремя строчками кода — проект `TextBoxSamples` (листинг 12.3).

Листинг 12.3. Автоматическая прокрутка в текстовом поле

```
'--------------------------------------
' Автоматическая прокрутка © 2004 Александр Климов
'--------------------------------------
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
        ' Добавить текст
        TextBox1.AppendText(" Добавленное слово")
        ' Установить выделение в конце текста и установить фокус
        TextBox1.SelectionStart = TextBox1.Text.Length
        TextBox1.Focus()
    End Sub
```

Аналогичный прием можно применить и к элементу управления `RichTextBox`.

12.3.2. Программный перевод фокуса на следующий или предыдущий элемент управления

Наверняка вам приходилось видеть программы, в которых после набора определенного числа символов в текстовом поле фокус автоматически переключался на другой элемент управления. Реализовать подобный подход можно через метод `Control.SelectNextControl()`. Предположим, при длине текста 7 символов нужно перевести фокус на следующее текстовое поле — проект `TextBoxSamples` (листинг 12.4).

Листинг 12.4. Перевод фокуса на другой элемент

```
    Private Sub txtNext1_TextChanged(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles txtNext1.TextChanged
        Const MAX_LENGTH As Integer = 7
        If txtNext1.Text.Length = MAX_LENGTH Then
```

```
            SelectNextControl(txtNext1, True, True, False, False)
        End If
    End Sub

    Private Sub txtNext2_TextChanged(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles txtNext2.TextChanged
        Const MAX_LENGTH As Integer = 7
        If txtNext2.Text.Length = MAX_LENGTH Then
            SelectNextControl(txtNext2, True, True, False, False)
        End If
    End Sub

    Private Sub txtNext3_TextChanged(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles txtNext3.TextChanged
        Const MAX_LENGTH As Integer = 7
        If txtNext3.Text.Length = MAX_LENGTH Then
            MsgBox("Спасибо")
            Close()
        End If
    End Sub
```

Как видите, при достижении длины в семь символов фокус от первого текстового поля переходит автоматически на второе текстовое поле. Потом все повторяется, и фокус переходит на третье текстовое поле.

12.3.3. Число строк в многострочном текстовом поле

Если у текстового поля свойство `Multiline` равно `True`, то свойство `Lines` возвращает массив строк в текстовом поле. Но у данного свойства есть небольшой недостаток — оно не учитывает перенос слова. На практике это выглядит следующим образом. Свойство `Lines` учитывает только строки, разделенные специальным символом переноса строки (константа `vbNewLine`). Но если у текстового поля свойство `WordWrap` установлено в `True`, то при длинных предложениях слова автоматически переносятся на новую строку. Свойство `Lines` никак не учитывает данное обстоятельство. И здесь нам пригодится функция Windows API с использованием сообщения `EM_GETLINESCOUNT`. Рассмотрим этот случай на конкретном примере в проекте `NumberLines`. Разместим на форме текстовое поле `TextBox1` и присвоим его свойству `Multiline` значение `True`. Также поместим кнопку `butCount` с надписью **Подсчитать** для подсчета строк в текстовом поле (листинг 12.5).

Листинг 12.5. Подсчет строк в текстовом поле

```
'--------------------------------------
' Подсчет строк в текстовом поле © 2004 Александр Климов
'--------------------------------------
    Const EM_GETLINECOUNT As Integer = &HBA
    Declare Function SendMessage Lib "user32" _
          Alias "SendMessageA" (ByVal hwnd As IntPtr, _
          ByVal wMsg As Integer, _
          ByVal wParam As Integer, _
          ByVal lParam As Integer) As Integer


    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        TextBox1.Text = "У лукоморья дуб зеленый;Златая цепь на дубе
        том:И днем и ночью кот ученый Все ходит по цепи кругом;
        Идет направо - песнь заводит, налево - сказку говорит."
    End Sub


    Private Sub butCount_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles butCount.Click
        ' число строк в тексте
        Dim NumberLines As Integer
        NumberLines = SendMessage(TextBox1.Handle _
                                  EM_GETLINECOUNT, 0, 0)
        MsgBox("Число строк: " & NumberLines)
        MsgBox(TextBox1.Lines.GetUpperBound(0))
    End Sub
```

Я намеренно поместил отрывок из поэмы А. С. Пушкина в одну строку, чтобы вы почувствовали разницу. Когда вы запустите проект, то увидите, что такой длинный текст не может поместиться на одной строчке и автоматически разобьется на несколько строк (свойство `WordWrap` должно быть `True`). Число получаемых строк будет зависеть от ширины текстового поля. Нажмите теперь на кнопку, и у вас появятся по очереди два сообщения. Первое сообщит о реальном количестве строк в текстовом поле, используя для подсчета строк сообщение Windows `EM_GETLINECOUNT`. Второе сообщение скажет, что в текстовом поле нет никаких строк, так как в искомом тексте нет ни одного символа перевода строки.

12.3.4. Блокировка контекстного меню текстового поля

У любого стандартного текстового поля есть контекстное меню, в которое входят команды **Отменить**, **Вырезать**, **Копировать**, **Вставить**, **Удалить**, **Выделить все**. Если в вашей программе необходимо блокировать это меню, то придется переопределять метод `WndProc`, обрабатывающий все сообщения Windows для текстового поля. Запустим новый проект `TextContextMenu`. Добавим в проект новый класс `DisabledContextMenu` и переопределим метод `WndProc` следующим образом (листинг 12.6).

Листинг 12.6. Блокировка контекстного меню

```
'---------------------------------------
' Блокировка контекстного меню © 2004 Александр Климов
' файл DisabledContextMenu.vb
'---------------------------------------
Public Class DisabledContextMenu
    Inherits TextBox
    Private Const WM_CONTEXTMENU As Integer = &H7B

    Protected Overrides Sub WndProc(ByRef m As
System.Windows.Forms.Message)
        If m.Msg <> WM_CONTEXTMENU Then
            MyBase.WndProc(m)
        End If
    End Sub
End Class
```

Напомню, что константы для сообщений Windows вы можете узнать из справочника по сообщениям Windows, размещенном на сайте **http://rusproject.narod.ru/guide.htm**.

Вернитесь в основную форму `Form1` и добавьте на нее два текстовых поля. Перейдите в редактор кода и разверните строчку `#Region " Windows Form Designer generated code "`. Найдите там следующие строчки:

```
Friend WithEvents TextBox1 As System.Windows.Forms.TextBox
Friend WithEvents TextBox2 As System.Windows.Forms.TextBox
```

Замените вторую строчку на следующую:

```
Friend WithEvents TextBox2 As DisabledContextMenu
```

Чуть ниже найдите строку:

```
Me.TextBox2 = New System.Windows.Forms.TextBox
```

Замените ее на такую:

```
Me.TextBox2 = New DisabledContextMenu
```

Запустив проект, попробуйте нажать правой кнопкой мыши на второе текстовое поле. Вы увидите, что никакого контекстного меню больше не появляется. Это означает, что текстовое поле фильтрует все поступающие к нему сообщения Windows и не пропускает для обработки заданное сообщение `WM_CONTEXTMENU`.

12.3.5. Запрет на комбинацию клавиш <Ctrl>+<X>

Совсем не обязательно блокировать контекстное меню полностью. Можно, например, установить запрет на определенную комбинацию клавиш. В частности, при выделенном в текстовом поле тексте комбинация клавиш <Ctrl>+<X> позволяет вырезать текст в буфер обмена (аналог команды **Вырезать** из контекстного меню). Попробуем переопределить метод `WndProc` таким образом, чтобы заблокировать данную комбинацию. Добавим в этот же проект новый класс `CtrlX` (листинг 12.7). Переопределение метода `WndProc` осуществим аналогично предыдущему примеру.

Листинг 12.7. Запрет на комбинацию клавиш <Ctrl>+<X> в текстовом поле

```
'--------------------------------------
' Запрет на комбинацию клавиш © 2004 Александр Климов
' файл CtrlX.vb
'--------------------------------------
Public Class CtrlX
    Inherits TextBox
    Private Const WM_CHAR As Integer = &H102

    Protected Overrides Sub WndProc(ByRef m As
System.Windows.Forms.Message)
        ' Если нажата клавиша CTRL
        If MyClass.ModifierKeys And Keys.Control Then
            Select Case m.Msg
                Case WM_CHAR
                    ' Блокируем CTRL+X.
                    Select Case m.WParam.ToInt32
```

```
                        Case 24 'X = 24 буква латинского алфавита
                            ' Ничего не делаем,
                            ' чтобы не обрабатывать сообщение
                        Case Else
                            MyBase.WndProc(m)
                    End Select
                Case Else
                    MyBase.WndProc(m)
            End Select
        Else
            MyBase.WndProc(m)
        End If
    End Sub
End Class
```

Добавляем на форму еще одно текстовое поле `TextBox3`. В редакторе кода делаем небольшие изменения:

```
' Закомментируем строчку и добавим свой вариант
'Friend WithEvents TextBox3 As System.Windows.Forms.TextBox
Friend WithEvents TextBox3 As CtrlX
...
Me.TextBox3 = New TextContextMenu.CtrlX
```

Запустите проект. В третьем текстовом поле попробуйте щелкнуть правой кнопкой мыши. Вы увидите, что контекстное меню работает как обычно. Попробуйте теперь выделить текст и вырезать его с помощью комбинации клавиш <Ctrl>+<X>. Эта комбинация работать не будет. Таким образом, мы успешно переопределили метод `WndProc`.

12.3.6. Только числа

На практике достаточно часто встречается задача ввода в текстовое поле только числовых значений. Существует множество реализаций данной задачи. Я хочу предложить вашему вниманию пример, который с успехом использовал в Visual Basic 6.0 при помощи встроенных функций Windows. У текстового поля существует стиль `ES_NUMBERS`, который позволяет блокировать ввод любых нечисловых символов. Но этого недостаточно для решения проблемы. Существует лазейка обойти это ограничение путем вставки текста из буфера. Поэтому нам понадобится также осуществлять проверку данных, находящихся в буфере обмена. Но пора перейти к примеру. Мы должны переопределить метод `WndProc` так, чтобы не допустить вставки из

буфера нечисловых данных. Как и в предыдущих примерах, добавим в проект DigitTextbox новый класс DigitOnly (листинг 12.8).

Листинг 12.8. Класс для ввода только чисел

```
'---------------------------------------
' Только числа © 2004 Александр Климов
' Файл DigitOnly.vb
'---------------------------------------
Imports System.Text.RegularExpressions
Imports System.Runtime.InteropServices
Public Class DigitOnly
    Inherits TextBox

    Public Sub New()
        ' Создадим шаблон для регулярного выражения
        ' Диапазон цифр от 0 до 9
        Me.regex = New Regex("[0-9]", RegexOptions.Compiled)
    End Sub
    ' Стиль для текстового поля. Разрешает ввод только числовых значений
    Private Const ES_NUMBER As Integer = &H2000

    ' Константа, используемая функциями буфера обмена
    Private Const CF_TEXT As Integer = 1

    ' Сообщение Windows
    Private Const WM_PASTE As Integer = &H302

    ' Регулярное выражение
    Private regex As regex

    ' Функции Windows API
    <DllImport("user32", SetLastError:=True)> _
        Private Shared Function OpenClipboard(ByVal hWndNewOwner As
        IntPtr) As Boolean
    End Function

    <DllImport("user32", SetLastError:=True)> _
```

```
Private Shared Function IsClipboardFormatAvailable(ByVal format As
Integer) As Boolean
End Function

<DllImport("user32", SetLastError:=True)> _
Private Shared Function GetClipboardData(ByVal format As Integer) As
IntPtr
End Function

<DllImport("user32", SetLastError:=True)> _
Private Shared Function CloseClipboard() As Boolean
End Function

<DllImport("kernel32", SetLastError:=True)> _
Private Shared Function GlobalLock(ByVal hMem As IntPtr) As IntPtr
End Function

<DllImport("kernel32", SetLastError:=True)> _
Private Shared Function GlobalUnlock(ByVal hMem As IntPtr) As Boolean
End Function

' Переопределяем метод WndProc
Protected Overrides Sub WndProc(ByRef m As
System.Windows.Forms.Message)
    Select Case m.Msg
        ' Обрабатываем сообщение вставки
    Case WM_PASTE
            ' Проверка формата данных в буфере обмена
            If IsClipboardFormatAvailable(CF_TEXT) Then
                ' Открываем буфер обмена
                If OpenClipboard(IntPtr.Zero) Then
                    ' Получим дескриптор буфера обмена в заданном
                      формате
                    Dim hTxt As IntPtr = GetClipboardData(CF_TEXT)

                    ' Фиксируем блок памяти
                    Dim hMem As IntPtr = GlobalLock(hTxt)

                    ' Работаем со строкой
                    Dim cliptxt As String =
                    Marshal.PtrToStringAnsi(hMem)
```

```
                        ' Освобождаем блок памяти
                        GlobalUnlock(hTxt)

                        ' Закрываем буфер обмена
                        CloseClipboard()

                        ' Сравниваем данные из буфера обмена с регулярным
                          выражением
                        If regex.Match(cliptxt).Success Then
                            'm.Result = IntPtr.Zero
                            MyBase.WndProc(m)
                        Else
                            'MyBase.WndProc(m)
                            m.Result = IntPtr.Zero
                        End If
                    End If
                Else
                    MyBase.WndProc(m)
                End If
            Case Else
                MyBase.WndProc(m)
        End Select
    End Sub

    ' Устанавливаем стиль Только Числа
    Protected Overrides ReadOnly Property CreateParams() As
System.Windows.Forms.CreateParams
        Get
            Dim cp As CreateParams = MyBase.CreateParams
            cp.Style = cp.Style Or ES_NUMBER
            Return cp
        End Get
    End Property
End Class
```

Переходим на главную форму проекта и добавляем два текстовых поля. В первом текстовом поле (`txtDigit`), которое будет являться полигоном для наших испытаний, очистим текст, а во втором (`txtAnyTxt`) оставим надпись по умолчанию `TextBox2`.

В редакторе кода делаем небольшие изменения:

```
Friend WithEvents txtDigit As DigitOnly
Me.txtDigit = New DigitTextbox.DigitOnly
```

Запустите проект. Попробуйте напечатать различные символы с клавиатуры в первом текстовом поле. Вы заметите, что текстовое поле не печатает буквенные символы и без проблем выводит числа. Переходим ко второй части испытаний. Скопируем в буфер обмена из второго текстового поля цифру 2 и попробуем вставить ее в первое текстовое поле (рис. 12.1). Никаких проблем. Теперь скопируйте часть текста и повторите операцию вставки. Вы увидите, что текстовое поле не принимает данные, содержащие буквы. Таким образом, мы создали вполне функциональный элемент управления с новыми свойствами.

В программе даны небольшие комментарии к коду программы, дополнительную информацию об используемых функциях Windows API вам нужно найти самостоятельно, чтобы лучше понять логику программы. Хочу чуть подробнее остановиться на используемом регулярном выражении. В качестве шаблона я использовал запись `[0-9]`, что включает в себя диапазон чисел от 0 до 9. В качестве альтернативы можно было использовать выражение `/d`. Но на мой взгляд первая запись более информативна и проще для запоминания.

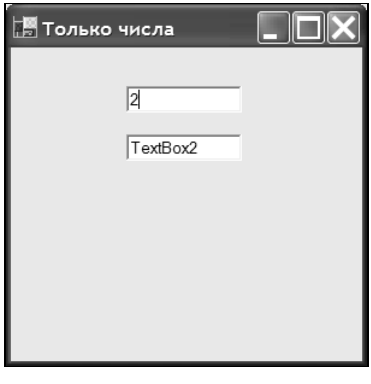

Рис. 12.1. Текстовое поле для ввода чисел

12.4. Надписи *Label*

12.4.1. Анимированные надписи

Вот еще один пример из моей коллекции кодов, написанных на VB 6.0. Идея кода состоит в следующем. На форме проекта `AniLabel` располагаем несколько надписей `Label` с различными размерами шрифта и цветом сим-

волов. Также поместим на форму кнопку butFirst с текстом Первый эффект и таймер Timer1. Затем с помощью таймера мы сжимаем надписи до минимальной величины и снова разжимаем их. В этот переходный период можно переопределить некоторые свойства надписей. Например, изменить текст надписей или цвет его символов. Вот код программы (листинг 12.9).

Листинг 12.9. Анимированные надписи

```
'--------------------------------------
' Анимированные надписи © 2004 Александр Климов
'--------------------------------------
    Dim A As Integer ' Шаг изменения Labels
    Dim LH As Integer ' Высота Label


Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        ' Вычисляем высоту Label и шаг изменения
        LH = Label1.Height
        A = LH / 10
    End Sub


    Private Sub butFirst_Click(ByVal sender As Object, ByVal e As
System.EventArgs) Handles butFirst.Click
        Timer1.Enabled = True
    End Sub


       Public Sub LabelMove(ByRef L As System.Windows.Forms.Label, ByRef
N As Short, ByRef H As Short)
        ' Если высота Label минус шаг изменения больше 0,
        ' то от высоты отнимаем шаг
        If L.Height - N > 0 Then
            L.Height = L.Height - N
            ' в противоположном случае
            ' изменяем знак шага на противоположный
        Else
            N = -N
            Label1.ForeColor = Color.Red
            Label2.Text = "Russian"
        End If


    End Sub
```

```
    Private Sub Timer1_Tick(ByVal sender As Object, ByVal e As
System.EventArgs) Handles Timer1.Tick
        ' Передаем поочередно Labels в процедуру обработки
        Call LabelMove(Label1, A, LH)
        Call LabelMove(Label2, A, LH)

        ' Выключаем таймер после того как Labels
        ' примут первоначальный размер
        If Label1.Height > LH Then
            Timer1.Enabled = False
            ' Изменяем значение шага на противоположное
            A = A * -1
        End If
    End Sub
```

Здесь я механически скопировал код из старого проекта. Единственное, что пришлось переделать, так это процедуру `LabelMove`. В VB 6.0 она выглядела так:

```
Public Sub LabelMove(L As Label, N As Integer, H As Integer)
```

Комментарии к программе достаточно понятно объясняют воспроизводимый эффект.

12.4.2. Переливающийся текст

Для создания переливающегося текста нам понадобится еще один таймер `Timer2` и кнопка `butSecond` с надписью `Второй эффект`, которые можно добавить в предыдущий проект `AniLabel`. Эффект переливающегося текста создается при плавном изменении интенсивности какого-нибудь цвета (листинг 12.10).

Листинг 12.10. Создание переливающегося текста

```
    Private Sub butSecond_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles butSecond.Click
        Timer2.Enabled = True
    End Sub

    Private Sub Timer2_Tick(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Timer2.Tick
        Static A As Integer
        A = A + 10 : If A > 510 Then A = 0
```

```
        ' Вы также можете изменить интервал 'Abs(A - 255)'
        ' Также вы можете изменить интервал таймера
        Label1.BackColor = Color.FromArgb(Math.Abs(A - 255), 0, 0)
        Label2.ForeColor = Color.FromArgb(Math.Abs(A - 255), 200, 100)
    End Sub
```

В приведенном примере я использовал старые возможности Visual Basic по представлению цвета. В VB 6.0 применялась упрощенная модель цвета, представленная сочетанием трех цветов: красного, зеленого и голубого. В новой версии языка была добавлена альфа-составляющая, которая позволяет управлять прозрачностью цвета. Воспользуемся этим обстоятельством для создания эффекта растворения текста. В этом случае достаточно вызвать другую перегруженную версию структуры `Color`:

```
' Для растворения текста на форме
' Label2.ForeColor = Color.FromArgb(Math.Abs(A - 255), 200, 200, 100)
```

Теперь будет меняться не составляющая цвета, а составляющая прозрачности цвета, что позволит при максимальном уровне прозрачности растворить текст на форме.

12.5. Меню

В VB .NET существуют два вида меню — `MainMenu` и `ContextMenu`. Их создание и работа с этими элементами обычно не вызывают затруднений у пользователя. Я расскажу, как отойти от привычной схемы и создавать нестандартные виды меню — с картинками, с различными начертаниями текста и с любыми цветами.

12.5.1. Создание цветных пунктов меню

У любого пункта меню есть свойство `OwnerDraw`. Если его установить в `True`, то это означает, что мы берем на себя всю работу по рисованию меню. В этом случае нужно написать код для событий `MeasureItem` и `DrawItem`. Опишем процесс с самого начала. Создаем новый проект `ColorMenu`. На панели задач находим элемент `MainMenu` и устанавливаем его на форме. В верхней части формы появится надпись `Type Here`. Введите следующее:

```
Необычное меню
```

Это будет текстом для верхнего уровня меню. Щелкните на нем мышью — сбоку и снизу снова появятся пункты `Type Here`. Выбираем нижний пункт и напечатаем, например, словосочетание `Цветной пункт`. Данный текст будет относиться к элементу `MenuItem2`. В редакторе свойств установите

для него свойство `OwnerDraw` в `True`. Можно приступать к написанию кода (листинг 12.11).

Листинг 12.11. Создание цветных пунктов меню

```
'--------------------------------------
' Цветное меню © 2004 Александр Климов
'--------------------------------------
    Private Const FONT_NAME As String = "Tahoma"
    Private Const FONT_SIZE As Single = 12
    Private Const FONT_STYLE As FontStyle = FontStyle.Bold

    Private Sub MenuItem2_MeasureItem(ByVal sender As Object, _
            ByVal e As System.Windows.Forms.MeasureItemEventArgs) Handles
MenuItem2.MeasureItem
        ' Создадим шрифт для вывода текста
        Dim menu_font As New Font( _
            FONT_NAME, FONT_SIZE, FONT_STYLE)

        ' Размеры текста
        Dim text_size As SizeF = _
            e.Graphics.MeasureString("Красный пункт", menu_font)

        ' Устанавливаем необходимые размеры для пункта меню
        e.ItemHeight = text_size.Height
        e.ItemWidth = text_size.Width
    End Sub

    Private Sub MenuItem2_DrawItem(ByVal sender As Object, ByVal e As
System.Windows.Forms.DrawItemEventArgs) Handles MenuItem2.DrawItem
        ' Создаем шрифт для вывода текста
        Dim menu_font As New Font( _
            FONT_NAME, FONT_SIZE, FONT_STYLE)

        ' Если мышь над пунктом меню
        If e.State And DrawItemState.Selected Then
            ' Draw a shaded background.
            Dim clr As Double
            Dim dclr As Double
            Dim Y As Long
```

```
            e.Graphics.FillRectangle( _
                    System.Drawing.Brushes.LightSkyBlue, _
                    e.Bounds.X, e.Bounds.Y, _
                    e.Bounds.Width, e.Bounds.Height)

            ' Выводим текст
            e.Graphics.DrawString("Другой текст", menu_font, _
                System.Drawing.Brushes.AliceBlue, _
                e.Bounds.X, e.Bounds.Y)
        Else
            ' Мышь за пределами пункта
            ' Меняем фон
            e.Graphics.FillRectangle( _
                System.Drawing.Brushes.Red, _
                e.Bounds.X, e.Bounds.Y, _
                e.Bounds.Width, e.Bounds.Height)

            ' Выводим текст
            e.Graphics.DrawString("Цветной пункт", menu_font, _
                System.Drawing.Brushes.Blue, _
                e.Bounds.X, e.Bounds.Y)
        End If
    End Sub
```

Обратите внимание, что можно менять не только фон для пункта меню, но и цвет для текста и сам текст пункта меню. Также отдельно настраиваются эти свойства, когда указатель мыши находится над нашим цветным пунктом меню.

12.5.2. Добавление картинки в меню

Кроме изменения свойств шрифта для пунктов меню можно также использовать стандартные картинки. На практике использование картинок в меню встречается достаточно редко, но небольшой пример на эту тему нам не помешает. Как и в предыдущем примере, используются два события — `MeasureItem` и `DrawItem` (проект `BitmapMenu`). Я добавил на форму графическое поле `PictureBox1`, в которое загрузил картинку с изображением кота. Эта картинка послужит нам образцом для создаваемого пункта меню. Далее создадим небольшое меню. Не забываем установить свойство `OwnerDraw` в

значение `True`, так как мы будем сами перерисовывать пункты меню (листинг 12.12).

Листинг 12.12. Добавление картинки в меню

```
'--------------------------------------
' Картинки в меню © 2004 Александр Климов
'--------------------------------------
    Private img As Bitmap

    Private Sub mnuBitmap_MeasureItem(ByVal sender As Object, ByVal e As
System.Windows.Forms.MeasureItemEventArgs) Handles mnuBitmap.MeasureItem
        e.ItemWidth = img.Width
        e.ItemHeight = img.Height
    End Sub

    Private Sub mnuBitmap_DrawItem(ByVal sender As Object, ByVal e As
System.Windows.Forms.DrawItemEventArgs) Handles mnuBitmap.DrawItem
        Dim rect As Rectangle = e.Bounds
        rect.X = e.Bounds.Width - img.Width
        rect.Width = img.Width
        e.DrawBackground()
        e.Graphics.DrawImage(img, rect)

    End Sub

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As Sys-
tem.EventArgs) Handles MyBase.Load
        img = PictureBox1.Image
    End Sub

    Private Sub mnuBitmap_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles mnuBitmap.Click
        MsgBox("Вы щелкнули на кошке!")
    End Sub
```

В начале нам надо сразу определить размеры для нашего пункта меню. Я просто присваиваю пункту размеры картинки:

```
e.ItemWidth = img.Width
e.ItemHeight = img.Height
```

Метод `DrawItem` рисует заданную картинку в нашем пункте меню. Запустите программу и щелкните на пункте **Меню**. У вас должен открыться подпункт с изображением кошки (рис. 12.2). Если вы щелкнете на картинке, то появится соответствующее сообщение.

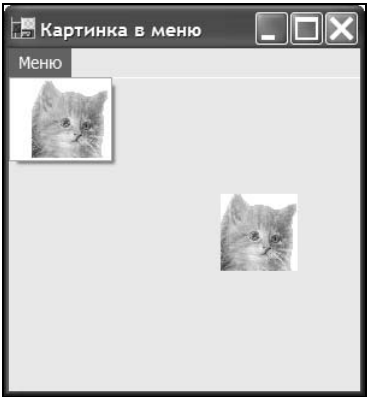

Рис. 12.2. Использование картинки в меню

12.5.3. Еще раз о цветном меню

В предыдущем примере было показано, как изменить цвет пунктов меню. Но у вас не получится изменить цвет самого меню, т. е. той самой полоски, которая находится сразу под заголовком, и на которой располагаются пункты верхнего меню. Но существует решение и этой проблемы. Как вы уже догадались, нам снова придется использовать функции Windows API (проект `ColorMainMenu`). Поместите на форму элемент `MainMenu` и создайте самостоятельно небольшую группу подменю. Для демонстрации примера они не нужны, поэтому особо не увлекайтесь (листинг 12.13).

Листинг 12.13. Создание цветной полосы меню

```
'--------------------------------------
' Цветное меню © 2004 Александр Климов
'--------------------------------------
    ' Структура и функции Windows API
    Public Structure MENUINFO
        Public cbSize As Int32
        Public fMask As Int32
        Public dwStyle As Int32
```

```
        Public cyMax As Int32
        Public hbrBack As IntPtr
        Public dwContextHelpID As Int32
        Public dwMenuData As Int32
        Public Sub New(ByVal owner As Control)
            cbSize = _
              System.Runtime.InteropServices.Marshal.SizeOf(_
                                          GetType(MENUINFO))
        End Sub
    End Structure

    Declare Function DrawMenuBar Lib "user32.dll" _
               (ByVal hwnd As IntPtr) As Int32
    Declare Function SetMenuInfo Lib "user32.dll" _
               (ByVal hmenu As IntPtr, _
                ByRef mi As MENUINFO) As Int32
    Declare Function CreateSolidBrush Lib "gdi32.dll" _
               (ByVal crColor As Int32) As IntPtr
    Declare Function DeleteObject Lib "gdi32.dll" _
                Alias "DeleteObject" (ByVal hObject As IntPtr) _
                As Boolean

    ' Константы
    Private Const MIM_BACKGROUND As Int32 = &H2
    Private hMenuBrush As IntPtr

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        ' Создаем цветную кисть
        hMenuBrush = CreateSolidBrush(RGB(250, 125, 65))

' Можно воспользоваться другим способом задания цвета
' hMenuBrush = CreateSolidBrush(ColorTranslator.ToOle(Color.Red))

    End Sub

    Private Sub Form1_Closed(ByVal sender As Object, ByVal e As
System.EventArgs) Handles MyBase.Closed
        ' Освобождаем ресурсы при закрытии формы
```

```
        DeleteObject(hMenuBrush)
    End Sub

    Private Sub Form1_Paint(ByVal sender As Object, ByVal e As
System.Windows.Forms.PaintEventArgs) Handles MyBase.Paint
        Dim mi As MENUINFO = New MENUINFO(Me)
        mi.fMask = MIM_BACKGROUND
        mi.hbrBack = hMenuBrush
        SetMenuInfo(Menu.Handle, mi)
        DrawMenuBar(Handle)
    End Sub
```

Пример работает следующим образом. Сначала вызывается функция `CreateSolidBrush` для создания новой кисти заданного цвета. Если вы больше привыкли к выбору цвета через структуру `Color`, то воспользуйтесь функцией `ToOle` класса `ColorTranslator`.

```
hMenuBrush = CreateSolidBrush(ColorTranslator.ToOle(Color.Red))
```

Получив дескриптор кисти выбранного цвета, передаем его функции `SetMenuInfo` в виде поля структуры `MENUINFO`. Функция `DrawMenuBar` перерисовывает изменившееся меню. Запустите проект, и вы увидите цветную полоску меню.

12.6. Переключатель *Radiobutton*

Переключатель иногда называют *радиокнопкой* — от сходства работы с кнопками на радиоприемниках. Только одна из этих кнопок-переключателей может быть отмечена точкой (свойство `Checked`). Таким образом, если в программе вам надо узнать, какая из кнопок имеет включенное состояние, то можно воспользоваться следующим примером — проект `RadioButtonSample` (листинг 12.14).

Листинг 12.14. Определение состояния переключателей

```
'--------------------------------------
' Переключатели © 2004 Александр Климов
'--------------------------------------
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
        Select Case True
            Case RadioButton1.Checked
```

```
            ' Здесь ваш код
            MsgBox("Вы выбрали RadioButton1")
        Case RadioButton2.Checked
            MsgBox("Вы выбрали RadioButton2")
        Case RadioButton3.Checked
            MsgBox("Вы выбрали RadioButton3")
    End Select
End Sub
```

Естественно, перед выполнением этого примера вам необходимо поместить на форму три переключателя `RadioButton` и одну обычную кнопку `Button`. Также не забудьте одному из переключателей присвоить значение `Checked = = True`.

12.7. Таймер

Элемент управления `Timer` очень прост и ненамного изменился по сравнению с предыдущей версией VB 6.0. Например, появились новые методы `Start` и `Stop`, эквивалентные установке свойств `Enabled` в `True` или `False`. Причем по умолчанию свойство `Enabled` установлено в `False` (раньше было `True`). Первое время меня это ставило в тупик. Запустив приложение, я не мог понять, почему ничего не происходит на экране. И только спустя некоторое время я вспоминал, что надо включить таймер. В примерах, разбросанных по страницам книги, достаточно часто используется таймер. Поэтому приведу здесь только один интересный пример использования данного элемента. В примере будет реализован принцип, применяемый некоторыми демо-версиями программ (проект `AutoClose`). Суть заключается в следующем — в течение определенного времени пользователь может ознакомиться с преимуществами программы. По истечении этого времени программа автоматически закрывается (листинг 12.15).

Листинг 12.15. Автоматическое закрытие программы

```
'--------------------------------------
' Автоматическое закрытие формы © 2004 Александр Климов
'--------------------------------------
    Private Sub Timer1_Tick(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Timer1.Tick
        ' Останавливаем таймер
        Timer1.Stop()
```

```
        ' Закрываем программу
        Close()
    End Sub
```

Вам только остается позаботиться об установке требуемого интервала для таймера.

12.8. Строка состояния *StatusBar*

Следующим объектом для наших опытов будет элемент управления `StatusBar` (строка состояния). У данного элемента нет встроенного свойства, позволяющего изменить цвет фона строки состояния и шрифт для выводимого в ней текста. Для реализации нашего желания придется самим заняться раскраской этого элемента. Запустим новый проект `ColorStatusBar`. Выберите на панели инструментов элемент `StatusBar` и поместите его на форму. Присвойте ему имя `sbColor`. В окне свойств для созданной строки состояния найдите пункт **Panels**. Щелкните на нем, а затем щелкните на кнопке с многоточием. У вас откроется диалоговое окно **StatusBarPanel Collection Editor**. Трижды нажмите на кнопку **Add**. В результате произведенных действий будет создано три панели с именами `StatusBarPanel1`, `StatusBarPanel2` и `StatusBarPanel3`. Измените свойство `Style` для каждой панели на `OwnerDraw` и нажмите кнопку **OK**, чтобы закрыть диалоговое окно. Снова зайдите в редактор свойств строки состояния `sbColor` и измените свойство `ShowPanels` на `True`. С визуальным программированием мы работать закончили. Переходим в редактор кода, где пишем весь необходимый код для перерисовки строки состояния (листинг 12.16).

Листинг 12.16. Создание цветной строки состояния

```
'--------------------------------------
' Цветная строка состояния © 2004 Александр Климов
'--------------------------------------
    Dim p As Pen = New Pen(Color.White)
    Dim sbr = New SolidBrush(Color.YellowGreen)
    Dim color_br(2) As SolidBrush
    Private Sub sbColor_DrawItem(ByVal sender As Object, _
     ByVal sbdevent As System.Windows.Forms.StatusBarDrawItemEventArgs) _
                                              Handles sbColor.DrawItem
        Dim g As Graphics = sbdevent.Graphics
        Dim sb As StatusBar = CType(sender, StatusBar)
        Dim rectf = New RectangleF(_
```

```
                        sbdevent.Bounds.X, sbdevent.Bounds.Y, _
                        sbdevent.Bounds.Width, sbdevent.Bounds.Height)

    g.DrawRectangle(p, sbdevent.Bounds)
    g.FillRectangle(color_br(sbdevent.Index), sbdevent.Bounds)
    g.DrawString("Панель" & sbdevent.Index, sb.Font, sbr, rectf)
End Sub

Private Sub Form1_Load(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles MyBase.Load
    color_br(0) = New SolidBrush(Color.White)
    color_br(1) = New SolidBrush(Color.Blue)
    color_br(2) = New SolidBrush(Color.Red)
End Sub

Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
    If disposing Then
        If Not (components Is Nothing) Then
            components.Dispose()
        End If

        p.Dispose()
        sbr.Dispose()
        Dim i As Integer
        For i = 0 To color_br.Length - 1
            color_br(i).Dispose()
        Next
    End If

    MyBase.Dispose(disposing)
End Sub
```

Запустив проект, вы увидите в строке состояния три панели, раскрашенные в цвета российского флага.

12.9. Элемент управления *ListView*

Элемент управления `ListView` предназначен для отображения списка различных элементов. Всем вам знаком Проводник Windows. Правая его часть

состоит из этого элемента управления. О том, как работать с этим элементом, вы сможете узнать из документации. Я расскажу о том, чего в документации нет. У элемента управления `ListView` есть такой элемент, как заголовок. Заголовок может включать не только текст, но и картинку, но встроенными средствами картинку в заголовок не поместить. Здесь приходится использовать неуправляемый код. Пример реализован в проекте `BitmapListView`. Разместите на форме элемент управления `ListView`, кнопку `Button` и список изображений `ImageList`. Все свойства установленных элементов управления оставляем без изменений. У элемента `ImageList` в редакторе свойств присвойте значению `Images` две любые картинки. Так как эти картинки будут использоваться в заголовках, то размеры у картинок должны быть не слишком большими. Код для помещения картинок в заголовки `ListView` выглядит так (листинг 12.17).

Листинг 12.17. Помещение картинок в заголовок элемента `ListView`

```
'---------------------------------------
' Картинки в ListView © 2004 Александр Климов
'---------------------------------------
Imports System.Runtime.InteropServices
    Public Const LVM_GETHEADER = 4127
    Public Const HDM_SETIMAGELIST = 4616
    Public Const LVM_SETCOLUMN = 4122
    Public Const LVCF_FMT = 1
    Public Const LVCF_IMAGE = 16
    Public Const LVCFMT_IMAGE = 2048

    ' Структура LVCOLUMN
    <StructLayout(LayoutKind.Sequential, pack:=8, CharSet:=CharSet.Auto)> _
    Structure LVCOLUMN
        Dim mask As Integer
        Dim fmt As Integer
        Dim cx As Integer
        Dim pszText As IntPtr
        Dim cchTextMax As Integer
        Dim iSubItem As Integer
        Dim iImage As Integer
        Dim iOrder As Integer
    End Structure
```

```
    ' Две разные перегруженные версии функции SendMessage
    ' Разница в последнем параметре
    <DllImport("User32.dll")> _
    Public Overloads Shared Function SendMessage _
        (ByVal hWnd As IntPtr, ByVal Msg As Integer, _
         ByVal wParam As Integer, ByVal lParam As Integer) As IntPtr
    End Function

    <DllImport("User32", CharSet:=CharSet.Auto)> _
    Public Overloads Shared Function SendMessage _
        (ByVal hWnd As IntPtr, ByVal msg As Integer, _
         ByVal wParam As Integer, ByRef lParam As LVCOLUMN) As IntPtr
    End Function

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        ListView1.View = View.Details
        ListView1.Columns.Add("Заголовок 0", ListView1.Width / 2, _
                                        HorizontalAlignment.Left)
        ListView1.Columns.Add("Заголовок 1", ListView1.Width / 2, _
                                        HorizontalAlignment.Left)
    End Sub

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
        Dim hwnd As IntPtr
        Dim iRet As IntPtr
        Dim i As Integer

        ' Получим дескриптор заголовка
        hwnd = SendMessage(ListView1.Handle, LVM_GETHEADER, 0, 0)

        ' Присоединим ImageList к заголовку
        iRet = SendMessage(hwnd, HDM_SETIMAGELIST _
                           0, (ImageList1.Handle).ToInt32)

        For i = 0 To ListView1.Columns.Count - 1
            Dim col As LVCOLUMN
            col.mask = LVCF_FMT Or LVCF_IMAGE
```

```
            col.fmt = LVCFMT_IMAGE
            ' Берем картинку из ImageList
            col.iImage = i
            col.cchTextMax = 0
            col.cx = 0
            col.iOrder = 0
            col.iSubItem = 0
            col.pszText = IntPtr.op_Explicit(0)

            ' Посылаем сообщение LVM_SETCOLUMN
            iRet = SendMessage(ListView1.Handle, LVM_SETCOLUMN, i, col)
        Next
    End Sub
```

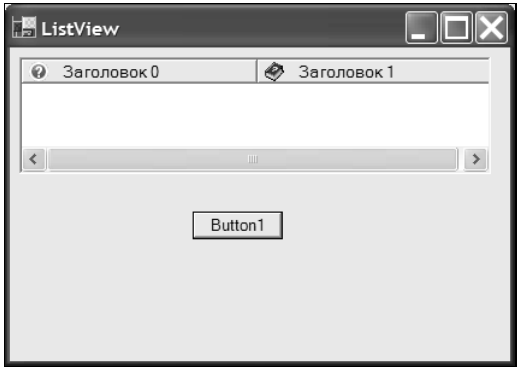

Рис. 12.3. Элемент `ListView` с картинками

12.10. Мигающий значок в области уведомлений

Раньше на форумах активно обсуждался вопрос: "Как поместить значок программы в область уведомлений?" Если быть точнее, применяли другое словосочетание: "поместить туда, где часики". Отчасти в таком определении области уведомлений виновата сама Microsoft, которая долгое время не давала официального перевода термина. На этих же форумах всех, кто задавал подобный вопрос, дружно посылали "читать специальный сборник часто задаваемых вопросов". Но не проходило и недели, как какой-нибудь новичок снова лез с таким вопросом. Все это заставило Microsoft сделать какие-то шаги в направлении, облегчающем жизнь программистам. И, наконец, в

версии VB .NET был предложен простой способ помещения значка в область уведомлений с помощью специально созданного элемента управления `NotifyIcon`. Порядок работы с этим элементом достаточно подробно описан в документации. Но простые примеры нам не интересны. Поэтому я предложу вам свой пример — мигающий значок. Первые шаги по построению примера стандартны. Создаем новый проект `BlinkNotify` и выбираем на панели инструментов два элемента: `Timer` и `NotifyIcon`. Обратите внимание, что они оба помещаются в специальную область, а не на форму. Установим сначала свойства:

- для таймера: `Interval` = `1000`;
- для элемента `NotifyIcon`:
  - `Icon` — выберите любой значок из вашей коллекции;
  - `Text` — введите какой-нибудь свой небольшой текст;
  - `Visible` = `False`.

Поместите на форму кнопку `Button1`, которая будет запускать таймер (листинг 12.18).

Листинг 12.18. Мигающий значок в области уведомлений

```
'-----------------------------------------
' Мигающий значок © 2004 Александр Климов
'-----------------------------------------
Me.Timer1.Start()

Private Sub NotifyIcon1_Click(ByVal sender As Object, ByVal e As
System.EventArgs) Handles NotifyIcon1.Click

        Me.Timer1.Stop()
        Me.NotifyIcon1.Visible = False
        MsgBox("Мигание приостановлено")
End Sub

Private Sub Timer1_Tick(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Timer1.Tick

        With Me.NotifyIcon1
            .Visible = (IIf(.Visible = True, False, True))
        End With
End Sub
```

Мигающий значок — весьма распространенный прием в программировании. Например, таким образом можно привлечь внимание пользователя при получении нового письма или сообщения интернет-пейджера типа ICQ.

12.11. Визуальные стили Windows XP

Теперь, когда вы познакомились с различными приемами работы с формой и элементами управления, есть смысл перейти к рассмотрению работы с визуальными стилями Windows XP. После выпуска операционной системы Windows XP пользователи увидели новый графический интерфейс. Элементы управления с закругленными углами, подсвечивание при прохождении мыши над объектом, необычная для глаз полоска индикатора. Многим такая разноцветная палитра пришлась по душе. Возник вопрос — как использовать всю эту красоту в своих приложениях? Имейте в виду, что дальнейшие объяснения относятся только к Windows XP. Сначала давайте сравним для наглядности две иллюстрации (рис. 12.4 и 12.5).

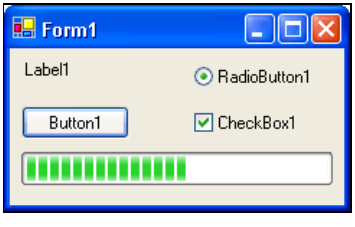

Рис.12.4. Стиль Windows XP

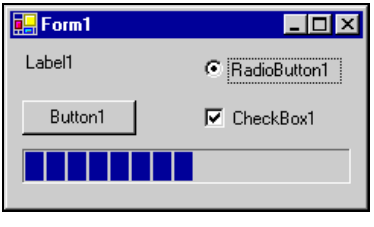

Рис. 12.5. Классический стиль

На рис. 12.4 вы видите новый вид элементов управления для Windows XP, на рис. 12.5 — элементы, используемые в Visual Basic.

Примечание

Как вы знаете, форма состоит из клиентской и неклиентской области. В неклиентскую область входят заголовок формы, ее границы и полосы прокрутки. Так вот эти элементы будут иметь стиль Windows XP без всякого кода с вашей стороны. А мы будем работать с элементами в клиентской области.

Новый внешний вид элементов управления связан с файлом Comctl32.dll версии 6.0. Вот список элементов управления, связанных с этим файлом и способных иметь новый стиль:

- `TextBox`;
- `RichTextBox`;
- `HscrollBar`;
- `ListView`;
- `TreeView`;
- `DateTimePicker`;

- VscrollBar;
- TrackBar;
- ComboBox;
- MonthCalendar;
- MainMenu;
- TreeView;
- ProgressBar;
- StatusBar;
- DataGrid;
- Splitter;
- ContextMenu;
- ListView;
- TabControl;
- ToolBar;
- ListBox.

Также в эту группу входят кнопочные элементы (`Button`, `RadioButton`, `GroupBox` и `CheckBox`), которые кроме связи с файлом Comctl32.dll должны также иметь стиль `System` в свойстве `FlatStyle`.

12.11.1. Метод *EnableVisualStyles*

Есть несколько способов заставить перечисленные элементы управления использовать визуальные стили Windows XP. Самый простой способ — применение встроенного метода `EnableVisualStyles` класса `Application`. Этот метод должен вызываться перед созданием элементов управления (как правило, в процедуре `Main`), а элементы управления должны иметь свойство `FlatStyle` равным `System` — проект `EnableStyles` (листинг 12.19). Наличие специального файла манифеста (о котором речь чуть далее) в этом случае необязательно.

Листинг 12.19. Использование метода `EnableVisualStyles`

```
'--------------------------------------
' Использование EnableVisualStyles © 2004 Александр Климов
'--------------------------------------
    Public Shared Sub Main()
        System.Windows.Forms.Application.EnableVisualStyles()
        ' Если возникают проблемы, то добавьте строчку
        Application.DoEvents()
        System.Windows.Forms.Application.Run(New Form1)
    End Sub
```

12.11.2. Использование манифеста

Но существует и второй способ. Если вы хотите использовать визуальные стили Windows XP в ваших приложениях, то должны добавить так называемый *манифест*, т. е. файл, показывающий системе, что приложение должно при возможности использовать файл Comctl32.dll шестой версии. Сама система Windows XP содержит две версии этого файла — пятую и шестую. По умолчанию система использует стандартную пятую версию. Таким образом,

чтобы использовать новые стили, мы должны как-то сообщить об этом системе. А как? Для этого и существует манифест. По своей сути манифест — это специальный XML-файл, который внедряется в приложение как ресурс или располагается отдельным файлом в одной папке с приложением.

Перейдем к практике. Создадим новый проект `Manifest`. Разместим на форме элементы управления: `Button`, `RadioButton`, `ProgressBar`, `CheckBox` и `Label`. Присвоим у всех кнопочных элементов свойству `FlatStyle` значение `System`. Добавим код для кнопки `Button`:

```
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
        ProgressBar1.Value = 50
    End Sub
```

Сохраните работу. Теперь приступим к написанию манифеста. В окне **Solution Explorer** щелкаем правой кнопкой мыши на имени проекта и выбираем команду **Add-New Item**. В появившемся диалоговом окне выбираем на левой панели (**Categories**) пункт **Local Project**. На правой панели (**Templates**) выбираем **Text File**. В текстовом поле пишем название файла следующим образом: `Имя проекта.exe.manifest`. Таким образом, если ваше приложение имеет имя WindowsApplication1, ваш XML-файл будет называться WindowsApplication1.exe.manifest. В нашем случае — Manifest.exe.manifest. Щелкните на кнопке **Open** (Открыть) для создания XML-файла и закрытия диалогового окна. Пустой текстовый файл добавится в текстовый редактор. Вставьте в этот текстовый файл строчки следующего содержания (листинг 12.20).

Листинг 12.20. Содержание файла манифеста

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<assembly xmlns="urn:schemas-microsoft-com:asm.v1" manifestVersion="1.0">
<assemblyIdentity
   version="1.0.0.0"
   processorArchitecture="X86"
   name="Microsoft.Winweb.<Executable Name>"
   type="win32"
/>
<description>.NET control deployment tool</description>
<dependency>
   <dependentAssembly>
     <assemblyIdentity
       type="win32"
```

```
        name="Microsoft.Windows.Common-Controls"
        version="6.0.0.0"
        processorArchitecture="X86"
        publicKeyToken="6595b64144ccf1df"
        language="*"
      />
   </dependentAssembly>
</dependency>
</assembly>
```

Замените строку `<Executable Name>` именем вашего приложения:

```
name="Microsoft.Winweb.Manifest
```

В меню **Build** выбираем команду **Build Manifest**, затем в меню **File** выбираем команду **Save All**. В папке проекта появится новый файл манифеста. Теперь необходимо подключить манифест к приложению. Переместите созданный файл Manifest.exe.manifest в одну папку с исполняемым файлом. Это может быть папка Debug или Release в папке obj в зависимости от ваших настроек. Запустив исполняемый файл, вы увидите ваши элементы управления с работающими визуальными стилями.

12.11.3. Манифест как ресурс

Можно добавить манифест в исполняемый файл как ресурс (проект `Resource`). Повторите все предыдущие шаги прошлого примера. Создайте простейший проект, поместив на форму кнопку и переключатель, а также создайте новый манифест для проекта. Теперь в меню **File** (Файл) выберите пункт **Open** (Открыть), затем щелкните на пункте **File** (Файл). Перейдите в папку, содержащую ваше решение. В этой папке откройте папку obj, затем папку Debug или Release (в зависимости от ваших настроек). Найдите в ней исполняемый файл (например, WindowsApplication1.exe) и дважды щелкните на нем. Данный файл откроется в интегрированной среде разработки Visual Studio. Щелкните правой кнопкой на исполняемом файле в дизайнере и выберите команду **Add Resource** (Добавить ресурс). У вас на экране появится диалоговое окно **Add Resource**. Щелкните на кнопке **Import** (Импортировать). Найдите созданный ранее файл манифеста. При поиске файла убедитесь, что в диалоговом окне у вас выбрано **All Files (*.*)** (Все файлы), иначе вы не найдете этот файл. Найдя манифест, дважды щелкните на нем. Откроется диалоговое окно **Custom Resource Type**. В поле **Resource Type** (Тип ресурса) введите `RT_MANIFEST` и щелкните на кнопке **OK**. В окне свойств установите свойство `ID` (идентификатор) равным 1. В меню **File** вы-

берите пункт **Save All** (Сохранить все). Запустив проект, вы увидите, что элементы управления поддерживают визуальные стили, хотя сам файл манифеста находится в ресурсах.

12.11.4. Проверка на использование визуальных стилей Windows XP

Если ваша программа активно использует визуальные стили Windows XP, то может возникнуть вопрос — поддерживает ли компьютер пользователя визуальные стили? Возможно, вы захотите изменить интерфейс и логику программы, если визуальные стили на компьютере отключены. Проверить, включена ли поддержка визуальных стилей Windows XP на данный момент можно с помощью функции Windows API `IsThemeActive`. Функция очень проста в использовании и не требует параметров. Кроме того, пользователь может в любой момент отказаться от поддержки визуальных стилей. Делается это следующим образом. Откройте Панель управления и запустите апплет **Экран**. На вкладке **Оформление** вы увидите выпадающий список **Окна и кнопки**. Здесь можно выбрать два варианта — **Стиль Windows XP** или **Классический вид**. В зависимости от вашего выбора и будет использоваться выбранный режим. Переключаться можно в любое время. Поэтому для отслеживания события переключения режима можно воспользоваться сообщением Windows. Для проекта `StyleXP` нам понадобятся одна кнопка `Button` и одна надпись `Label` (листинг 12.21).

Листинг 12.21. Проверка использования визуальных стилей Windows XP

```
'--------------------------------------
' Визуальные стили Windows XP © 2004 Александр Климов
'--------------------------------------
    Declare Function IsThemeActive Lib "uxtheme.dll" () As Boolean
    Private Const WM_THEMECHANGED As Int32 = &H31A

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
        Label1.Text = IsThemeActive.ToString
    End Sub

    Protected Overrides Sub WndProc(ByRef m As
System.Windows.Forms.Message)
        Select Case m.Msg
            Case WM_THEMECHANGED
```

```
            Label1.Text = "Тема изменилась"
        End Select
        MyBase.WndProc(m)
    End Sub
```

Запустив программу, попробуйте теперь поменять тему указанным выше способом. Как только Windows получит сообщение о смене темы, в вашем текстовом поле появится соответствующая надпись. Также вы можете определять через кнопку, поддерживаются ли на данный момент визуальные стили Windows XP.

Примечание

В будущей версии .NET Framework 2.0 и Visual Basic .NET 2005 для определения активности тем планируется ввести новое свойство `IsEnabledByUser` класса `System.Windows.Forms.VisualStyles.VisualStyleInformation`.

12.12. Советы и рекомендации

- Задание к *разд. 12.1* — попробуйте для закрепления материала создать произвольной формы не только кнопку, но и другие элементы управления — например, надпись `Label`.
- Задание к *разд. 12.2* — примените использованный прием к другим элементам управления.
- Обратите внимание, что текстовое поле позволяет использовать цифровые символы. Если же вам понадобится ввести дробное число, то у вас возникнут проблемы со знаком разделителя. Возможно, вам придется полностью переписывать пример для решения проблемы.

ЧАСТЬ V

"Продвинутые" примеры

Глава 13

Шутки и розыгрыши

В этой главе будут представлены программы шутливого характера. Несмотря на некоторую несерьезность примеров, изучение подобных упражнений позволит глубже понять структуру операционной системы Windows, принципы взаимодействия объектов, процессы в работе мыши и клавиатуры. Единственное, на что хотелось бы обратить внимание читателя, — не следует злоупотреблять своими знаниями во вред пользователю. Шутки должны носить легкий, веселый характер и служить для поднятия настроения. Не стоит писать пакостные программы, которые могут всерьез напугать не очень опытного пользователя.

13.1. Скрытие и показ курсора

В качестве первого примера для шутки продемонстрируем возможность скрытия на время курсора мыши. Будьте рядом со своим другом, которому вы подложите эту свинью. Поначалу он будет судорожно двигать мышью по коврику, полагая, что ее указатель вышел за пределы экрана. Для усиления эффекта можно постепенно затемнять форму и, в конце концов, вывести на экран предупреждающее сообщение в духе самой Windows: **Программа выполнила недопустимую операцию. Требуется срочно обновить драйвер для коврика мышки!** Реализация этого примера очень проста. У класса `Cursor` есть метод `Hide`, который и прячет курсор мыши. Так как данный прием действует в пределах формы, то для большей выразительности можно развернуть форму на весь экран — проект `Mouse` (листинг 13.1).

Листинг 13.1. Скрытие и показ курсора мыши

```
'--------------------------------------
' Шутки с мышью © 2004 Александр Климов
'--------------------------------------

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
```

```
        Cursor.Hide()
    End Sub

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
        Cursor.Show()
    End Sub
```

Использовать метод `Hide` надо осторожно, так как он обладает накопительным свойством. Если вы два раза вызовете метод `Hide`, то вам придется дважды вызывать и метод `Show`, восстанавливающий курсор мыши.

13.2. Замена кнопок мыши

У вашей мыши имеются, как минимум, две кнопки. Если вы правша, то основная кнопка у вас левая. Правая кнопка служит тогда для вызова контекстного меню. Но у левшей своя точка зрения на эти кнопки. И основная кнопка у них правая. Эти настройки доступны через Панель управления в апплете **Мышь**. Но вы можете программно изменить эти настройки — проект `Mouse` (листинг 13.2).

Листинг 13.2. Замена кнопок мыши

```
    Declare Function SwapMouseButton Lib "user32.dll" _
       Alias "SwapMouseButton" (ByVal bSwap As Integer) As Integer

    Private Sub butSwap_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles butSwap.Click
        ' Меняем местами левую и правую кнопку мыши
        SwapMouseButton(1)
    End Sub
```

Эффект получается очень неожиданный. Сила привычки настолько велика, что даже зная, что кнопки поменялись местами, все равно постоянно путаешься с ними. Чтобы вернуть прежнее поведение мыши, нужно вызвать эту же функцию с другим параметром:

```
SwapMouseButton(0)
```

13.3. Ни минуты покоя

Указатель мыши можно не только прятать, но и перемещать по экрану с помощью свойства `Cursor.Position`. Поместите на форму кнопку и таймер.

В событии таймера заставим бегать указатель по всему экрану, используя случайные числа — проект `MadMouse` (листинг 13.3).

Листинг 13.3. Перемещение мыши по случайным координатам

```
'--------------------------------------
' Беспокойная мышка © 2004 Александр Климов
'--------------------------------------
    Private Sub Timer1_Tick(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Timer1.Tick
        ' Случайное число от 1 до 799
        Dim m_Value As Integer
        m_Value = CInt(Int((799 * Rnd()) + 1))

        ' Случайное число от 1 до 599
        Dim m_Value2 As Integer
        m_Value2 = CInt(Int((599 * Rnd()) + 1))

        Randomize()
        Cursor.Position = New Point(m_Value, m_Value2)
    End Sub

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
        ' Останавливаем таймер
        Timer1.Stop()
    End Sub
```

В этом примере каждые полсекунды указатель мыши перемещается на случайную позицию в прямоугольнике 800×600 пикселов. Если вы установите слишком маленький интервал для таймера, то будет затруднительно закрыть программу мышью. Для этого случая я разместил на форме кнопку. Так как на форме это единственный элемент управления, то она получит фокус при загрузке формы. И у вас будет возможность остановить таймер нажатием клавиш пробела или <Enter>.

13.4. Мышеловка

Совсем мы замучили бедную мышку. Но и на этом не остановимся. Если у вас завелась мышь, а любимый кот отказывается ее ловить, то приходится

использовать мышеловку. Для создания виртуальной мышеловки нам понадобится маленькая кнопочка — проект `MouseTrap` (листинг 13.4).

Листинг 13.4. Создание виртуальной мышеловки

```
'--------------------------------------
' Мышеловка © 2004 Александр Климов
'--------------------------------------


    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
        ' Чтобы освободить мышь,
        ' используйте комбинацию клавиш Ctrl+Alt+Del
        Cursor.Clip = RectangleToScreen(_
                      New Rectangle(Button1.Location, Button1.Size))
    End Sub
```

Запустите проект и попросите пользователя нажать на кнопку. Зря он это сделал. Мышка попалась в мышеловку! Теперь курсор мыши не сможет выйти за пределы кнопки. Причем ограничивающий прямоугольник будет действовать и в других программах. Единственный способ освободить мышь — нажать комбинацию из трех клавиш <Ctrl>+<Alt>+<Del>. Интересно, пользователь догадается об этом?

13.5. Убегающая кнопка

Очень эффектно смотрится пример с убегающей кнопкой. Выглядит это следующим образом. На форме находится обычная кнопка с дразнящей надписью. Однако при попытке подвести курсор мыши к кнопке она вдруг перепрыгивает на другое место. Лично мне так и не удалось нажать на нее. Может быть вам повезет больше. Для создания подобной шутки нам понадобится стандартная форма с единственной кнопкой с надписью **Нажми меня**. Эффект реализуется через код для события `MouseMove` — проект `RunawayButton` (листинг 13.5).

Листинг 13.5. Пример с убегающей кнопкой

```
'--------------------------------------
' Убегающая кнопка © 2004 Александр Климов
'--------------------------------------
```

```
    Private Sub Button1_MouseMove(ByVal sender As Object, ByVal e As
System.Windows.Forms.MouseEventArgs) Handles Button1.MouseMove
        Button1.Left = Rnd() * (ClientSize.Width - Button1.Width)
        Button1.Top = Rnd() * (ClientSize.Height - Button1.Height)
    End Sub
```

Можно запустить проект и посмотреть на результат. Однако у нашего примера есть один "баг". Дело в том, что на кнопку можно нажать не только мышью, но и с помощью клавиатуры. Запустите еще раз пример и нажмите на клавишу пробела. Удерживая клавишу нажатой, попробуйте подвести мышь к кнопке. Кнопка начнет беспорядочно метаться по форме. Для устранения такого эффекта добавим перед написанным кодом еще несколько строчек:

```
If (e.X < 0) Or (e.Y < 0) Or _
   (e.X > Button1.Width) Or _
   (e.Y > Button1.Height) Then _
Exit Sub
```

Идея, заложенная в примере, очень проста. Как только курсор мыши подходит к кнопке, то возникает событие `MouseMove`. И программа сразу же перемещает кнопку в другое случайное место с помощью функции `Rnd`. Таким образом, пользователь просто не успевает щелкнуть на кнопке. Обратите внимание, что в данном примере мы используем функцию `Rnd`, унаследованную от старой VB 6.0. В версии VB .NET появился общий для .NET Framework класс `Random`, которым пользоваться предпочтительнее, если вы в будущем собираетесь писать примеры, скажем, на C#.

13.6. Выбор не за вами!

В предыдущем примере вы не могли щелкнуть на кнопке, потому что она убегала. Но существует и другой способ, чтобы не позволить пользователю щелкнуть на кнопке. Этот способ основан на возможности программно перемещать курсор мыши в нужное место. Пусть на форме будут две кнопки с надписями **Да** и **Нет**. Как только мы подведем мышь к кнопке **Нет**, то курсор сразу переместится в середину кнопки с надписью **Да**. И у вас не останется выбора, кроме как щелкнуть на кнопке, на которую, возможно, так не хотелось нажимать. Реализация задуманного эффекта выглядит следующим образом. Разместите на форме надпись `Label`. Придумайте текст для надписи `Label1` самостоятельно. В качестве примера я выбрал вопрос: **Вы хотите посетить сайт http://rusproject.narod.ru/ еще раз?** Под созданной надписью располагаем две кнопки `Button`. Одной из них присваиваем в качестве надписи на кнопке слово **Да**, другой — слово **Нет**. Естественно, мне, как автору

сайта, не хотелось бы, чтобы вы выбрали вариант со словом **Нет**. Пишем код для второй кнопки — проект `TwoButtons` (листинг 13.6).

Листинг 13.6. Принудительное перемещение курсора

```
'---------------------------------------
' Отсутствие выбора © 2004 Александр Климов
'---------------------------------------
    Private Sub Button2_MouseMove(ByVal sender As Object, ByVal e As
System.Windows.Forms.MouseEventArgs) Handles Button2.MouseMove
        Dim startPoint As Point = Button1.PointToScreen(New Point(0, 0))
        startPoint.X += (Button1.Width / 2)
        startPoint.Y += (Button1.Height / 2)
        Cursor.Position = startPoint
    End Sub
```

Мы написали код для обработчика события `MouseMove`. Данное событие начинает обрабатываться, как только курсор мыши достигает границ кнопки. В этом случае сразу же вычисляется местоположение другой кнопки, и курсор мыши автоматически перемещается в подготовленное место. Обратите внимание, что для вычисления координат кнопки мы используем метод `PointToScreen`, который конвертирует координаты из клиентской системы координат в экранную систему координат. К этому трюку приходится прибегать из-за используемого свойства курсора `Position`, которое работает только с экранными координатами. На самом деле этот пример можно использовать не только в качестве шутки. Иногда требуется автоматически перенаправить курсор мыши в нужное место. Думаю, будет полезным оформить пример в виде универсальной процедуры. Разместим на форме еще один элемент управления, например, гиперссылку `LinkLabel`. В редакторе кода пишем код для универсальной процедуры (листинг 13.7).

Листинг 13.7. Универсальная процедура перемещения мыши

```
Public Sub JumpToControl(ByVal ctrl As Control)
        ' Перемещаем курсор мыши в нижнюю часть середины элемента
        Dim objPoint As Point = ctrl.PointToScreen(New Point(0, 0))
        objPoint.X += (ctrl.Width / 2)
        objPoint.Y += ((ctrl.Height / 4) * 3)
        Cursor.Position = objPoint
End Sub
```

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
        JumpToControl(LinkLabel1)
End Sub
```

Как видите, код для процедуры практически идентичен примеру с кнопками. Теперь достаточно указать имя элемента управления, на который должен перемещаться указатель мыши. Обратите внимание, что для процедуры мы выбрали перемещение не в середину элемента управления, а на одну треть его высоты.

13.7. Панель задач

Особенно интересно смотрятся шутки, связанные с привычными элементами интерфейса Windows. Все мы привыкли, что в нижнем левом углу рабочего стола у нас находится кнопка **Пуск** (при стандартном расположении панели задач). Кроме кнопки **Пуск** на панели задач также можно увидеть кнопки запущенных программ, область уведомлений (в правом углу), системные часы. Все эти элементы являются по своей сути обычными окнами. А значит, получив доступ к такому окну, можно изменить привычное поведение элемента. Правда, для достижения намеченной цели встроенных классов .NET Framework уже недостаточно. Придется призывать на помощь функции Windows API. В нашем случае нам понадобятся две функции: `FindWindowEx` и `ShowWindow`. Объявления этих функций в Visual Basic .NET выглядят следующим образом:

```
Private Declare Auto Function FindWindowEx Lib "user32.dll" ( _
        ByVal hwnd As IntPtr, _
        ByVal hWndChild As IntPtr, _
        ByVal lpszClassName As String, _
        ByVal lpszWindow As String _
    ) As IntPtr

Private Declare Auto Function ShowWindow Lib "user32.dll" ( _
        ByVal hwnd As IntPtr, _
        ByVal nCmdShow As Int32 _
    ) As Int32
```

Также нам понадобятся две константы, отвечающие за показ и скрытие окна:

```
Private Const SW_HIDE As Int32 = 0
Private Const SW_SHOW As Int32 = 5
```

Теперь мы можем с помощью указанных ранее функций Windows API найти дескриптор нужного окна по имени его класса. Вот небольшой список имен классов элементов, присутствующих на панели задач:

- `Shell_TrayWnd` — панель задач;
- `Button` — кнопка **Пуск**;
- `TrayNotifyWnd` — область уведомлений со значками и часами;
- `TrayClockWClass` — часы;
- `SysTabControl32` — кнопки запущенных программ.

13.7.1. Издевательства над кнопкой *Пуск*

Итак, мы выяснили, что кнопка **Пуск** тоже является обычным окном со своими свойствами. Наша задача — получить доступ к этому окну и изменить его поведение. А, имея доступ к кнопке, можно придумать множество шутливых издевательств над пользователем.

Скрытие кнопки *Пуск*

В этом примере мы покажем, как программно получить доступ к кнопке и скрыть ее от пользователя с экрана монитора (проект `Pusk`). Поместим на форму две кнопки. Одна кнопка будет отвечать за скрытие кнопки **Пуск**, вторая — за показ этой кнопки на экране (листинг 13.8).

Листинг 13.8. Скрытие кнопки Пуск

```
'--------------------------------------
' Примеры с кнопкой Пуск © 2004 Александр Климов
'--------------------------------------
    ' Функции Windows API
    Private Declare Auto Function FindWindowEx Lib "user32.dll" ( _
            ByVal hwnd As IntPtr, _
            ByVal hWndChild As IntPtr, _
            ByVal lpszClassName As String, _
            ByVal lpszWindow As String _
        ) As IntPtr

    Private Declare Auto Function ShowWindow Lib "user32.dll" ( _
            ByVal hwnd As IntPtr, _
            ByVal nCmdShow As Int32 _
        ) As Int32
```

```
    Private Declare Function MoveWindow Lib "user32" Alias "MoveWindow" _
            (ByVal hwnd As IntPtr, ByVal x As Integer, _
            ByVal y As Integer, _
            ByVal nWidth As Integer, ByVal nHeight As Integer, _
            ByVal bRepaint As Integer) As Integer

    Private Declare Function GetWindowRect Lib "user32" _
           (ByVal hWnd As IntPtr, ByRef lpRect As RECT) As Integer

    Private Declare Function SendMessage Lib "user32" _
            Alias "SendMessageA" _
           (ByVal hwnd As IntPtr, _
            ByVal wMsg As Integer, ByVal wParam As Integer, _
            ByVal lParam As Integer) As Integer

    ' Структуры
    Private Structure RECT
        Public Left As Integer
        Public Top As Integer
        Public Right As Integer
        Public Bottom As Integer
    End Structure

    ' Константы
    Private Const SW_HIDE As Int32 = 0
    Private Const SW_SHOW As Int32 = 5
    Private Const WM_SYSCOMMAND As Int32 = &H112
    Private Const SC_TASKLIST As Long = &HF130

    ' Дескриптор кнопки Пуск
    Private hWndStart As IntPtr
    ' Дескриптор области уведомлений
    Private tray As IntPtr

    Private Sub butHide_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles butHide.Click
        ' Ищем окно с классом Shell_TrayWnd
        Dim hW As IntPtr = FindWindowEx(IntPtr.Zero, _
               IntPtr.Zero, "Shell_TrayWnd", vbNullString)
```

```
        ' Получим дескриптор кнопки Пуск
        hWndStart = FindWindowEx(hW, IntPtr.Zero, "BUTTON", vbNullString)
        ' Прячем кнопку
        ShowWindow(hWndStart, SW_HIDE)
    End Sub
```

Разберем пример поподробнее. Сначала объявим глобальную переменную `hWndStart`, в которой будет содержаться дескриптор кнопки **Пуск**. При нажатии кнопки с надписью **Спрятать Пуск** в переменную `hW`, заносится значение дескриптора панели задач, возвращаемое функцией Windows API `FindWindowEx` через указанный класс `Shell_TrayWnd`.

Кнопка **Пуск** принадлежит классу `BUTTON` и является дочерним окном панели задач. Поэтому следующая строчка находит дескриптор кнопки по имени класса:

```
hWndStart = FindWindowEx(hW, IntPtr.Zero, "BUTTON", vbNullString)
```

Функция `ShowWindow` служит для различных операций над окном: скрытия, показа, восстановления, сворачивания, развертывания и т. д. В нашем случае требуется скрыть кнопку с помощью константы `SW_HIDE`. Но нам необходимо иметь возможность восстановить кнопку на экране, иначе она так и останется невидимой до следующей перезагрузки Windows! Поэтому необходимо иметь вторую кнопку, где мы пишем:

```
    Private Sub butShow_Click(ByVal sender As Object, ByVal e As
System.EventArgs) Handles butShow.Click
        ' Показать кнопку Пуск
        ShowWindow(hWndStart, SW_SHOW)
    End Sub
```

Благодаря тому, что мы использовали глобальную переменную `hwndStart` для дескриптора кнопки, нам уже не нужно второй раз получать его через вызов функции `FindWindowEx`, и мы сразу вызываем функцию `ShowWindow` с константой `SW_SHOW` (как альтернативу можно использовать константу `SW_RESTORE`). Естественно, имея доступ к окну кнопки **Пуск**, можно не только скрывать его, но проделывать и другие операции, допустимые с окнами.

Перемещение кнопки *Пуск*

Имея дескриптор окна (в данном случае кнопки **Пуск**), мы можем перемещать окно в другое место. Для решения этой задачи нам опять понадобятся функции Windows API `MoveWindow` и `GetWindowRect`, а также структура `RECT`. Функция `GetWindowRect` получает размеры и координаты окна, а функция `MoveWindow` непосредственно перемещает окно в указанное место. Мы уже знаем, как получить дескриптор кнопки. Осталось получить размеры и ме-

стоположение кнопки и переместить кнопку в новую позицию. Добавим в проект две новые кнопки butMoveRight и butMoveLeft. Первая кнопка будет сдвигать кнопку **Пуск** на 100 пикселов вправо (рис. 13.1), а вторая — возвращать кнопку **Пуск** на свое законное место. Сначала посмотрим, как переместить кнопку **Пуск** вправо (листинг 13.9).

Листинг 13.9. Смещение кнопки Пуск вправо

```
    Private Sub butMoveRigth_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles butMoveRigth.Click
        Dim r As RECT
        ' Получим размеры кнопки Пуск
        GetWindowRect(hWndStart, r)
        ' Сдвигаем кнопку вправо
        MoveWindow(hWndStart, r.Left + 100, 0, _
           r.Right - r.Left, r.Bottom - r.Top, True)
    End Sub
```

Здесь с помощью функции GetWindowRect мы получаем информацию о позиции и размере кнопки **Пуск**, которая записывается в структуру RECT. Теперь достаточно правильно распорядиться полученной информацией. Я рекомендую вам не смещать кнопку вдоль оси *Y* и не менять ее высоту, так как полученный результат выглядит не слишком красиво. Но вы можете попробовать увеличить ширину кнопки. Пользователь будет в шоке! Чтобы вернуть кнопку на место, установите прежние координаты кнопки (листинг 13.10).

Листинг 13.10. Возвращение кнопки Пуск на место

```
    Private Sub butMoveLeft_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles butMoveLeft.Click
        Dim r As RECT
        GetWindowRect(hWndStart, r)
        MoveWindow(hWndStart, 0, 0, _
           r.Right - r.Left, r.Bottom - r.Top, True)
    End Sub
```

На основе этих примеров видно, что можно достичь очень интересных результатов. Например, у меня есть программы для своих друзей, в которых кнопка **Пуск** периодически подпрыгивает вверх и падает вниз, самопроизвольно нажимается в самый неожиданный момент (об этом — чуть далее), меняет свою надпись на заданную строчку. Попробуйте самостоятельно до-

биться таких же результатов. А если придумаете какое-нибудь новое издевательство над кнопкой, то не поленитесь и пришлите свой пример на мой электронный адрес.

Рис. 13.1. Перемещение кнопки **Пуск**

Программное нажатие на кнопку *Пуск*

Еще один способ напугать пользователя — это неожиданно программно нажать на кнопку **Пуск**. Можно сделать это с использованием таймера. Я вам покажу только заготовку для примера (листинг 13.11).

Листинг 13.11. Программное нажатие на кнопку Пуск

```
    Private Sub butPress_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles butPress.Click
        ' Нажимаем на кнопку Пуск
        SendMessage(Me.Handle, WM_SYSCOMMAND, SC_TASKLIST, 0)
    End Sub
```

Сначала надо объявить функцию Windows API `SendMessage` и сообщение `WM_SYSCOMMAND` с константой `SC_TASKLIST`. Теперь можно в любом месте кода вызвать эту функцию, что приводит к открытию меню кнопки **Пуск**.

13.7.2. Область уведомлений

Для закрепления материала я предлагаю расширить пример. Давайте попробуем скрыть область уведомлений. *Область уведомлений* — это та часть панели задач, где находятся часы, значок раскладки клавиатуры, значки некоторых программ. Технология нам уже знакома по предыдущему примеру. Объявляем глобальную переменную `tray`, в которой будет содержаться дескриптор области уведомлений. Дальнейший код практически совпадает с кодом скрытия кнопки **Пуск** (листинг 13.12).

Листинг 13.12. Скрытие и показ области уведомления

```
    Private Sub butHideTray_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles butHideTray.Click
        Dim hW As IntPtr = FindWindowEx(IntPtr.Zero, _
                IntPtr.Zero, "Shell_TrayWnd", vbNullString)
```

```
        ' Дескриптор области уведомлений
        tray = FindWindowEx(hW, IntPtr.Zero, _
               "TrayNotifyWnd", vbNullString)
        ' Прячем область уведомлений
        ShowWindow(tray, SW_HIDE)
    End Sub

Private Sub butShowTray_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles butShowTray.Click
        ShowWindow(tray, SW_SHOW)
End Sub
```

Системные часы

Мы только что рассмотрели пример скрытия области уведомлений. Вероятно, вы поспешили самостоятельно попытаться скрыть системные часы, тем более, что я уже назвал имя его класса — `TrayClockWClass`. Не торопитесь, у вас ничего не получится. Здесь кроется одна небольшая хитрость. Дело в том, что системные часы не являются дочерним окном панели задач, как вы могли подумать. Их родительское окно — область уведомлений. Поэтому нам придется дважды вызывать функцию `FindWindowEx`, чтобы получить дескриптор окна с часами (листинг 13.13).

Листинг 13.13. Скрытие и показ системных часов

```
    Private Sub butClockHide_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles butClockHide.Click
        ' Дескриптор панели задач
        Dim hW As IntPtr = FindWindowEx(IntPtr.Zero, IntPtr.Zero, _
                                        "Shell_TrayWnd", vbNullString)
        ' Дескриптор области уведомлений
        tray = FindWindowEx(hW, IntPtr.Zero, "TrayNotifyWnd", _
                            vbNullString)
        ' Дескриптор системных часов
        Dim trayclock = FindWindowEx(tray, IntPtr.Zero, _
                            "TrayClockWClass", vbNullString)
        ' Прячем системные часы
        ShowWindow(trayclock, SW_HIDE)
    End Sub
```

```
    Private Sub butClockShow_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles butClockShow.Click
        Dim hW As IntPtr = FindWindowEx(IntPtr.Zero, _
                   IntPtr.Zero, "Shell_TrayWnd", vbNullString)
        tray = FindWindowEx(hW, IntPtr.Zero, _
                   "TrayNotifyWnd", vbNullString)
        Dim trayclock = FindWindowEx(tray, IntPtr.Zero, _
               "TrayClockWClass", vbNullString)
        ShowWindow(trayclock, SW_SHOW)
    End Sub
```

Замена системных часов на бегущую строку

Скрытие системных часов — слишком простая задача. Гораздо интереснее заменить эти часики на что-нибудь свое. Я покажу вам, как можно заменить системные часы на бегущую строку (проект `TrayClockReplace`). Для начала надо установить свойство формы `FormBorderStyle` в значение `None`. После этого поместите на форму надпись `Label` и таймер `Timer`. Включите таймер установкой свойства `Enabled` в `True` и задайте интервал, равный 10 миллисекундам. Все остальные свойства элементов управления будут задаваться программным путем (листинг 13.14).

Листинг 13.14. Замена системных часов на бегущую строку

```
'--------------------------------------
' Замена системных часов © 2004 Александр Климов
'--------------------------------------
    Structure RECT
        Public Left As Integer
        Public Top As Integer
        Public Right As Integer
        Public Bottom As Integer
    End Structure

    Declare Auto Function FindWindowEx Lib "user32.dll" ( _
            ByVal hwnd As IntPtr, _
            ByVal hWndChild As IntPtr, _
            ByVal lpszClassName As String, _
            ByVal lpszWindow As String _
        ) As IntPtr
```

```
    Declare Function GetWindowRect Lib "user32" _
           (ByVal hWnd As IntPtr, ByRef lpRect As RECT) _
            As Integer

    Declare Function SetParent Lib "user32" _
           (ByVal hWndChild As IntPtr, _
            ByVal hWndNewParent As IntPtr) As Integer

    Private Const sText As String = "Сообщение  "

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load

        Dim r As RECT

        ' Дескриптор панели задач
        Dim hShell As IntPtr = FindWindowEx(IntPtr.Zero, IntPtr.Zero, _
                                        "Shell_TrayWnd", vbNullString)

        ' Дескриптор области уведомлений
        Dim tray As IntPtr = FindWindowEx(hShell, IntPtr.Zero, _
                                      "TrayNotifyWnd", vbNullString)

        ' Дескриптор системных часов
        Dim trayclock = FindWindowEx(tray, IntPtr.Zero,
"TrayClockWClass", vbNullString)

        ' Размеры часов
        GetWindowRect(trayclock, r)

        ' Установим размеры формы
        With Me
            .Top = 1
            .Left = 1
            .Height = r.Bottom - r.Top - 1
            .Width = r.Right - r.Left - 1
        End With
```

```
        ' Поместим форму в область часов
        SetParent(Me.Handle, trayclock)
    End Sub

    Private Sub Timer1_Tick(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Timer1.Tick
        Label1.Text = sText
        If Label1.Left + Label1.Width < 0 Then Label1.Left = Me.Width
        Label1.Left = Label1.Left - 1
    End Sub

    Private Sub Label1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Label1.Click
        Close()
    End Sub

    Private Sub Form1_Resize(ByVal sender As Object, ByVal e As
System.EventArgs) Handles MyBase.Resize
        Label1.Top = (Me.Height - Label1.Height) / 2
    End Sub

    Private Sub Form1_DoubleClick(ByVal sender As Object, ByVal e As
System.EventArgs) Handles MyBase.DoubleClick
        Close()
    End Sub
```

Запустите проект. Вы должны увидеть, что вместо системных часов в нижнем правом углу экрана у вас появилась бегущая строка с сообщением.

13.8. Открыть и закрыть CD-ROM

Одна из самых популярных шуток среди программистов — это периодическое открывание и закрывание привода CD-ROM. В одном из своих давних интервью Касперский (руководитель антивирусной компании "Лаборатория Касперского") сказал, что это, пожалуй, единственный пример вируса, действительно выводящего из строя "железо" компьютера, а не установленное програмное обеспечение. Представьте себе, что вирус начнет ночью выдвигать и задвигать лоток CD-ROM с интервалом в 1 секунду (существуют пользователи, которые даже ночью оставляют компьютер включенным). Ресурс любого "железа" имеет свой предел, и не исключено, что к утру ваш привод будет уже неработающим. Мы, конечно, не будем ломать наши дис-

ководы. Но знание такого приема может пригодиться, например, при написании своего музыкального проигрывателя. Для реализации данной задачи требуется всего одна функция Windows API `mciSendString` — проект `OpenCD` (листинг 13.15).

Листинг 13.15. Открытие лотка CD-ROM

```
'---------------------------------------
' Открытие лотка CD-ROM © 2004 Александр Климов
'---------------------------------------
   Private Declare Function mciSendString Lib "winmm.dll _
                                  Alias "mciSendStringA" _
                                 (ByVal lpstrCommand As String, _
                                  ByVal lpstrReturnString As String, _
                                  ByVal uReturnLength As Long, _
                                  ByVal hwndCallback As Long) As Long


    Private Sub butOpenCD_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles butOpenCD.Click
        ' Откроем лоток CD-ROM
        mciSendString("set CDAudio door open", 0, 127, 0)
    End Sub


    Private Sub butCloseCD_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles butCloseCD.Click
        ' Закроем лоток
        mciSendString("set CDAudio door closed", 0, 127, 0)
    End Sub
```

Как видите, всего две строчки кода, позволяют написать весьма эффектную программу.

13.9. Блокировка клавиатуры и мыши

Однако давайте снова вернемся к мышам. Можно не прятать мышь, а просто на время заблокировать ее. Заодно заблокируем и клавиатуру. Представляете себе испуг пользователя, судорожно пытающего сдвинуть с места курсор мыши. Для этого примера достаточно вызвать функцию Windows API `BlockInput`. Так как мы не садисты, то давайте заблокируем мышь и клавиатуру всего на три секунды. Разместим на форме кнопку, при нажатии на

которую и будет происходить указанное действо — проект `BlockAll` (листинг 13.16).

Листинг 13.16. Блокировка мыши и клавиатуры

```
'---------------------------------------
' Блокировка мыши и клавиатуры © 2004 Александр Климов
'---------------------------------------
     Declare Function BlockInput Lib "User32" _
           (ByVal fBlockIt As Boolean) As Boolean


    Private Sub butBlock_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles butBlock.Click
        ' Начало блокировки клавиатуры и мыши
        BlockInput(True)
        ' Пауза на 3 секунды
        Me.Text = "Заблокировано на 3 секунды"
        System.Threading.Thread.Sleep(3000)
        ' Снимаем блокировку
        BlockInput(False)
        Me.Text = "Блокировка снята"
    End Sub
```

Теперь при нажатии на кнопку блокируются любые попытки нажать на кнопки мыши и клавиши клавиатуры. Единственный способ прервать блокировку — это нажать одновременно комбинацию из трех клавиш <Ctrl>+ +<Alt>+<Del>. Но не всякий пользователь догадается нажать эти клавиши.

13.10. Печатающая машинка

Во многих фантастических фильмах рассказывается об ужасах, происходящих после выхода умных машин из-под контроля человека. Представьте себе, что ваша клавиатура, служившая вам верой и правдой несколько лет, захотела независимости и стала сама печатать без вашего участия! Хотите посмотреть как это выглядит? Тогда за дело. Создаем новый проект `TypingMachine` и размещаем на форме одну кнопку и один таймер. Так как кнопка не несет никакой смысловой нагрузки, то я не стал менять ее имя, только присвоил свойству `Text` значение `Старт`. У таймера я присвоил свойству `Interval` значение `400`, остальные свойства оставил по умолчанию. Кнопка служит всего лишь для запуска таймера и скрытия самой формы.

Основные события разворачиваются в обработчике события `Tick` таймера `Timer1` (листинг 13.17).

Листинг 13.17. Печатающая машинка

```
'--------------------------------------
' Печатающая машинка © 2004 Александр Климов
'--------------------------------------
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
        Me.Hide()
        Timer1.Enabled = True
    End Sub


     Private Sub Timer1_Tick(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Timer1.Tick
        Dim a As Integer
        Randomize()
        a = Rnd() * 100 + 1
        'If a < 45 Then SendKeys.SendWait(Chr(Rnd() * 26))
        If a > 45 And a < 70 Then SendKeys.SendWait("Кто сказал мяу?
{Left 16} Гав!")
        If a > 70 And a < 80 Then SendKeys.SendWait("{enter 2} Мяу
мяу{enter}")
        If a = 72 Then Timer1.Enabled = False
        'If a > 80 And a < 93 Then SendKeys.SendWait("{BS}")
        'If a > 93 And a < 99 Then SendKeys.SendWait("^{BS}")
        'If a > 99 Then SendKeys.SendWait("+{HOME}{BS}")
    End Sub
```

Я пожалел вашу психику и выбрал щадящий режим. Запустите сначала программу Блокнот (notepad.exe), а затем сам проект. Когда нажмете на кнопку **Старт**, программа спрячется, и фокус перейдет на открытый заранее Блокнот. Так как таймер по-прежнему работает и посылает команды клавиатуре, в Блокноте будут печататься различные слова. Но класс `SendKeys` содержит очень богатые возможности. Ведь можно печатать не просто символы, но и эмулировать нажатия таких клавиш, как <Del>, <BackSpace>, <Alt>, <Ctrl> и т. д. Я закомментировал несколько таких строчек. Для пущего эффекта запустите несколько разных приложений, в которых возможен ввод символов. Снимите символы комментариев со строк кода и запустите программу. У вас начнется форменная свистопляска. У меня данная

программа умудрилась за несколько секунд очистить содержимое нескольких файлов, переименовать их и даже удалить!

Примечание

Пожалуйста, будьте осторожны при использовании этого примера! Закройте все важные документы и программы. Предусмотрите возможность принудительного закрытия программы. Используйте программу на свой страх и риск!

13.11. Отгадыватель мыслей

Впервые эту шутку я увидел на сайте **http://www.arbuz.uz**. Всем любителям арбузов и занимательной математики рекомендую посетить этот сайт. Там вы найдете много статей, написанных в очень доступной форме.

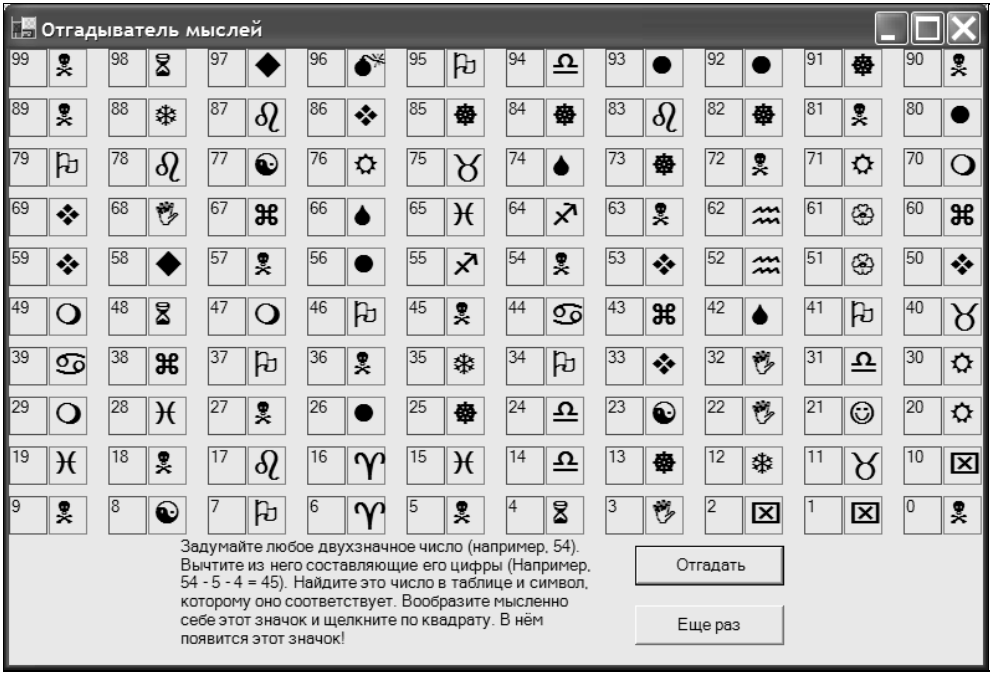

Рис. 13.2. Отгадыватель мыслей

Часто статьи сопровождаются листингами программ на Visual Basic, Pascal и Java. Но вернемся к отгадывателю мыслей. После опубликования на сайте эта страничка стала очень известной. Количество посетителей страницы исчислялось тысячами в день. Все пытались узнать, как же удается программе

угадать мысли человека. Мне удалось подобрать специальный алгоритм, который сработает и в бумажном варианте. Задумайте любое двухзначное число (например, 54). Вычтите из него составляющие его цифры (например, 54 − 5 − 4 = 45). Найдите это число в таблице и символ, которому оно соответствует (рис. 13.2).

Вообразите этот значок себе мысленно. А теперь взгляните на рис. 13.3, и вы увидите задуманный символ!

Рис. 13.3. Задуманный символ

Вот как выглядит эта программа на Visual Basic .NET (проект `Mind`). Для программы нам понадобятся две надписи и две кнопки. В первой надписи будет отображаться информация о том, как пользоваться программой. Во второй будет появляться задуманная картинка (листинг 13.18). При размещении на форме ее следует сделать невидимой (`Visible = False`).

Листинг 13.18. Программа отгадывания мыслей

```
'---------------------------------------
' Отгадыватель мыслей © 2004 Александр Климов
'---------------------------------------
    ' Массив строк
    Dim alphaArray() As String = {"a", "b", "d", "f", "h", "{", "i", "l",
"v", "x", "z", "I", "J", "M", "N", "o", "O", "R", "S", "T", "U", "m",
"6", "^", "u", "_", "[", "]"}
    ' Случайное число
    Dim MyValue As Integer


    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
        Label1.Visible = True


        Dim g As Graphics = CreateGraphics()
        g.Clear(Me.BackColor)
    End Sub


    Private Sub Form1_Paint(ByVal sender As Object, ByVal e As
System.Windows.Forms.PaintEventArgs) Handles MyBase.Paint
        Button2.PerformClick()
    End Sub
```

```
    Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button2.Click
        Dim i, y As Integer
        Dim g As Graphics = CreateGraphics()
        g.Clear(BackColor)
        Label1.Visible = False
        Dim MyValue As Integer
        MyValue = CInt(Int(20 * Rnd()))

        For i = 0 To 9
            For y = 0 To 9
                Dim to9 As Integer
                to9 = 99 - ((i) + (y * 10))

                g.DrawRectangle(Pens.Red, i * 80, y * 40, 30, 30)

                g.DrawString(Convert.ToString(to9) _
                    Me.Font, New SolidBrush(Color.Black), i * 80, y * 40)

                g.DrawRectangle(Pens.Gray, i * 80 + 32, y * 40, 30, 30)

                Dim randnum As Random
                randnum = New Random

                Dim f As Font = New Font("wingdings", 24, _
                                FontStyle.Bold, GraphicsUnit.Pixel)

                If to9 Mod 9 = 0 Then
                    g.DrawString(alphaArray(MyValue), f, _
                      New SolidBrush(Color.Black), i * 80 + 32, y * 40)
                    Label1.Text = alphaArray(MyValue)
                ElseIf to9 Mod 9 = 1 Then
                    g.DrawString(alphaArray(randnum.Next(28)), f, _
                        New SolidBrush(Color.Black), i * 80 + 32, y * 40)
                Else
                    g.DrawString(alphaArray(Int((28 * Rnd()))), f _
                        New SolidBrush(Color.Black), i * 80 + 32, y * 40)
                End If
```

```
            Next
        Next
        g.Dispose()

    End Sub
```

Как вы уже, наверное, догадались, программа и не пытается угадать ваши мысли. Любители занимательной математики без труда поймут, как программа отгадывает выбранный вами рисунок. Я не стану раскрывать секрет на страницах книги. Попробуйте найти ответ самостоятельно. Впрочем, после изучения кода программы, это не составит большого труда. Программа начинается с объявления массива `alphaArray` и переменной `MyValue`. В целях экономии ресурсов, чтобы не использовать картинки, программа использует символы из шрифта Wingdings. В этот набор входит много интересных символов с изображением различных картинок. Часть этих символов мы и будем использовать в качестве загадки, поместив их в массив `alphaArray`. Переменная `MyValue` в случайном порядке выбирает один из этих символов. При загрузке формы выполняется код из кнопки `Button2` (метод `PerformClick`). В событии `Click` кнопки `Button2` заключен основной код программы. Здесь рисуются 99 пар прямоугольников с помощью вложенных циклов. В первых прямоугольниках каждой пары выводятся их порядковые номера от 0 до 99. В оставшихся прямоугольниках выводятся в случайном порядке символы из набора шрифта Wingdings. Так как здесь хранится разгадка секрета программы, то больше ничего объяснять не буду!

13.12. Искусственный интеллект

Недавно у меня состоялся интересный диалог с моим компьютером. Я задавал ему самые разные вопросы на животрепещущие темы, а он отвечал мне. Вот небольшой отрывок из нашего диалога:

— Кто ты?

— Твой папочка.

— Ты уверен?

— Наверное.

— Как тебя зовут?

— Нехороший вопрос.

— Когда я разбогатею?

— Ты будешь слишком стар, когда это будет.

— А кто будет тогда президентом?

— Ты не думаешь, что это Владимир Жириновский?

— Ты не врешь?

— Все знают, что да.

— Зачем ты врешь?

— Я всего лишь машина.

А ваш компьютер способен отвечать на ваши вопросы? Если нет, то я помогу вам создать подобную программу. Для проекта `AI` нам понадобятся кнопка `Button`, надпись `Label`, текстовое поле `TextBox` и список `ListBox`. Сначала код программы (листинг 13.19).

Листинг 13.19. Создание искусственного интеллекта

```
'--------------------------------------
' Искусственный интеллект © 2004 Александр Климов
'--------------------------------------
   Function OldQuestion(ByRef UseText As String) As Boolean
        Dim i As Integer
        For i = 0 To lstUsed.Items.Count - 1
            If UCase(lstUsed.Items(i)) = UCase(UseText) Then
                OldQuestion = True
                Exit Function
            Else
                OldQuestion = False
            End If
        Next
    End Function

    Function OkQuestion(ByRef TheText As String) As String
        Dim TempText As String
        Dim Ekstra As String
        Dim Text(9) As String
        Dim Number As Short

        If InStr(1, TheText, "elvis", CompareMethod.Text) Then GoTo
        Theking

        TempText = Replace(TheText, " ", "")
        If TheText = TempText Then
            Ekstra = ""
```

```
        Else
            Ekstra = " "
        End If
Start:
        If InStr(1, TheText, "Что" & Ekstra, CompareMethod.Text) Then
GoTo WhichWhatHow
        If InStr(1, TheText, "Как" & Ekstra, CompareMethod.Text) Then
GoTo WhichWhatHow
        If InStr(1, TheText, "Где" & Ekstra, CompareMethod.Text) Then
GoTo Where
        If InStr(1, TheText, "Почему" & Ekstra, CompareMethod.Text) Then
GoTo Why
        If InStr(1, TheText, "Который" & Ekstra, CompareMethod.Text) Then
GoTo WhichWhatHow
        If InStr(1, TheText, "Кто" & Ekstra, CompareMethod.Text) Then
GoTo Who
        If InStr(1, TheText, "Когда" & Ekstra, CompareMethod.Text) Then
GoTo When_Renamed

        Text(0) = "Я думаю так"
        Text(1) = "А ты что думаешь?"
        Text(2) = "Да"
        Text(3) = "Нет"
        Text(4) = "Все знают, что нет"
        Text(5) = "Наверно"
        Text(6) = "Не уверен"
        Text(7) = "Да, конечно."
        Text(8) = "Нет"
        Text(9) = "Все знают, что да"
        Number = Int((Rnd() * 10) + 1) - 1
        Return Text(Number)
        Exit Function
WhichWhatHow:
        Text(0) = "Откуда мне знать?"
        Text(1) = "Я устал - поговорим завтра"
        Text(2) = "Эхх, мне бы твои заботы"
        Text(3) = "Ну, как тебе сказать"
        Text(4) = "Любопытный вопрос ... кто-то уже спрашивал"
        Text(5) = "Нехороший вопрос"
        Text(6) = "Ты хочешь знать, что я думаю об этом?"
        Text(7) = "Я не хочу говорить об этом"
```

```
        Text(8) = "Извини, забыл"
        Text(9) = "Терпеть не могу любопытных людей"
        Number = Int((Rnd() * 10) + 1) - 1
        Return Text(Number)
        Exit Function
Where:
        Text(0) = "Может во Флориде?"
        Text(1) = "Я думаю, это где-то в Америке."
        Text(2) = "Подожди...Я пороюсь в атласе."
        Text(3) = "В Германии"
        Text(4) = "Прямо на небесах"
        Text(5) = "На скотном дворе"
        Text(6) = "Сам догадайся"
        Text(7) = "В Белом доме"
        Text(8) = "Последний раз я его видел в нижнем белье"
        Text(9) = "Может в стиральной машине?"
        Number = Int((Rnd() * 10) + 1) - 1
        Return Text(Number)
        Exit Function
Why:
        Text(0) = "Потому что кончается на У, дурачок!"
        Text(1) = "Открой энциклопедию!"
        Text(2) = "Это судьба"
        Text(3) = "Гм, я не знаю"
        Text(4) = "А я так хочу!"
        Text(5) = "Спроси Монику Левински"
        Text(6) = "Ну, ты попроще задавай вопросы"
        Text(7) = "Я не помню"
        Text(8) = "Поясни вопрос"
        Text(9) = "А почему ты не спросишь об этом маму?"
        Number = Int((Rnd() * 10) + 1) - 1
        Return Text(Number)
        Exit Function
Who:
        Text(0) = "Ты не думаешь, что это Владимир Жириновский"
        Text(1) = "Бонд, Джеймс Бонд"
        Text(2) = "Джим Кэрри"
        Text(3) = "Чужой"
```

```
        Text(4) = "Тетя Соня"
        Text(5) = "Врач-гинеколог"
        Text(6) = "Дед в кожаном пальто"
        Text(7) = "Твой папочка"
        Text(8) = "Билл Клинтон"
        Text(9) = "Памела Андерсон"
        Number = Int((Rnd() * 10) + 1) - 1
        Return Text(Number)
        Exit Function
When_Renamed:

        Text(0) = "Завтра"
        Text(1) = "Вчера"
        Text(2) = "Это было в 1900"
        Text(3) = "Ты будешь слишком стар, когда это будет"
        Text(4) = "Давно это было"
        Text(5) = "Во время гражданской войны"
        Text(6) = "Когда ты родился"
        Text(7) = "Когда ты первый раз побрился"
        Text(8) = "Ну, это было когда Билл Клинтон и Моника Левински..."
        Text(9) = "Сейчас"
        Number = Int((Rnd() * 10) + 1) - 1
        Return Text(Number)

        Exit Function
Theking:

        If InStr(1, TheText, "alive", CompareMethod.Text) Or InStr
        (1, TheText, "living", CompareMethod.Text) Or InStr(1, TheText,
        "dead", CompareMethod.Text) Then

            Return "А ты как думал ? Конечно жив, Он сейчас король ;)"
        Else
            GoTo Start
        End If
    End Function

    Function NoQuestion() As Object
        Dim Text(9) As String
```

```
        Dim Number As Short
        Text(0) = "Ну, попробуй подумать"
        Text(1) = "Я лучше отвечу на вопросы"
        Text(2) = "А может лучше задашь вопросик"
        Text(3) = "Я люблю вопросы"
        Text(4) = "Пожалуйста, задай мне вопрос"
        Text(5) = "Я всего лишь машина .... ну, спроси меня"
        Text(6) = "Попробуй еще раз задать вопрос"
        Text(7) = "OK, но сейчас задай вопрос"
        Text(8) = "Попробуй задать мне вопрос"
        Text(9) = "Я только отвечаю"
        Randomize()
        Number = Int((Rnd() * 10) + 1) - 1
        NoQuestion = Text(Number)
    End Function

    Function AI(ByRef Text As String) As String

        Dim TempText As String
        Dim Ekstra As String
        TempText = Replace(Text, " ", "")
        If OldQuestion(TempText) = False Then
            lstUsed.Items.Add(TempText)

            If Text = TempText Then
                Ekstra = ""
            Else
                Ekstra = " "
            End If
            If InStr(1, Text, "Что" & Ekstra, CompareMethod.Text) Then
GoTo Question
            If InStr(1, Text, "Как" & Ekstra, CompareMethod.Text) Then
GoTo Question
            If InStr(1, Text, "Где" & Ekstra, CompareMethod.Text) Then
GoTo Question
            If InStr(1, Text, "Почему" & Ekstra, CompareMethod.Text) Then
GoTo Question
            If InStr(1, Text, "Который" & Ekstra, CompareMethod.Text)
Then GoTo Question
```

```
            If InStr(1, Text, "Кто" & Ekstra, CompareMethod.Text) Then
GoTo Question
            If InStr(1, Text, "Когда" & Ekstra, CompareMethod.Text) Then
GoTo Question
            If Microsoft.VisualBasic.Right(TempText, 1) = "?" Then GoTo
Question
            Return NoQuestion()
            Exit Function
Question:
            Return OkQuestion(Text)
        Else
            Return "Я уже отвечал на это."
        End If
    End Function
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
        Label1.Text = AI(TextBox1.Text)
    End Sub
```

Основой для программы послужил старый код, написанный еще на Visual Basic 5.0. Я постарался ничего не менять в старой программе, поэтому вы увидите множество устаревших конструкций. А суть программы заключается в обработке первого слова вопроса. Например, если в вопросе содержится слово `Почему`, то ответ программа выбирает случайным образом из небольшого набора предложений в разделе `Why` функции `OkQuestion`. Здесь представлена только заготовка для создания программы с искусственным интеллектом. Но если творчески подойти к составлению ответов, то программа может доставить вам немало веселых минут.

13.13. Я вижу все!

А сейчас я хочу предложить вашему вниманию еще одну забавную программу. В этой программе глаза кошки будут постоянно наблюдать за перемещениями мыши по экрану. В какой точке экрана не находилась бы мышь, глаза обязательно будут направлены в эту точку — проект `Spycat` (листинг 13.20).

Листинг 13.20. Слежение за указателем мыши

```
'--------------------------------------
' Кот, следящий за мышкой © 2004 Александр Климов
'--------------------------------------
```

```
    ' Позиция указателя мыши
    Private CursorPos As Point

    ' Координаты левого и правого глаза в экранных координатах
    Private LeftEye As Point
    Private RightEye As Point

    ' Расстояние между глазом и курсором
    Private LeftDistance As Double
    Private RightDistance As Double

    ' Синусы и косинусы углов от глаз до указателя мыши
    Private LeftSin As Single
    Private LeftCos As Single
    Private RightSin As Double
    Private RightCos As Double

    ' Координаты глаз в координатах формы
    Private Const LX As Long = 60         'координата по оси x
    Private Const RX As Long = LX + 24    'координата по оси x
    Private Const BY As Long = 60         'координата по оси y

    ' Диаметр зрачка
    Private Const PupilRad As Single = 6
    ' Диаметр движения зрачка
    Private Const MoveRad As Single = 3
    ' Диаметр глаз
    Private Const EyeRad As Single = 12

    Private Sub Timer1_Tick(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Timer1.Tick

        ' Вычисляем позицию глаз в экранных координатах
        LeftEye = PointToScreen(New Point(LX, BY))
        RightEye = PointToScreen(New Point(RX, BY))

        ' Получаем позицию указателя мыши
        CursorPos = Cursor.Position
```

```
' вычисляем расстояния
LeftDistance = _
  Math.Sqrt((LeftEye.X - CursorPos.X) ^ 2 + _
  (LeftEye.Y - CursorPos.Y) ^ 2)

RightDistance =  _
  Math.Sqrt((RightEye.X - CursorPos.X) ^ 2 + _
  (RightEye.Y - CursorPos.Y) ^ 2)

If LeftDistance = 0 Then ' на 0 делить нельзя
    LeftDistance = 1
End If

If RightDistance = 0 Then ' на 0 делить нельзя
    RightDistance = 1
End If

' вычисляем синус и косинус угла
LeftSin = (LeftEye.Y - CursorPos.Y) / LeftDistance
LeftCos = (LeftEye.X - CursorPos.X) / LeftDistance

RightSin = (RightEye.Y - CursorPos.Y) / RightDistance
RightCos = (RightEye.X - CursorPos.X) / RightDistance

Dim g As Graphics = CreateGraphics()
g.Clear(BackColor)

' Рисуем мордочку кота
g.FillEllipse(Brushes.Gold, LX - 15, BY - 15, 60, 60)

' ушки на макушке
Dim aptLEar() As Point = _
  {New Point(50, 60), New Point(40, 25), New Point(64, 49)}
g.FillPolygon(Brushes.Gold, aptLEar)

Dim aptREar() As Point = _
  {New Point(86, 47), New Point(105, 29), New Point(97, 56)}
g.FillPolygon(Brushes.Gold, aptREar)
```

```
        ' Носик и ротик
        g.FillEllipse(Brushes.Black, 72, 72, 8, 8)
        g.DrawArc(Pens.Black, 66, 71, 20, 20, 30, 120)

        ' усы
        g.DrawLine(Pens.White, 77, 83, 135, 71)
        g.DrawLine(Pens.White, 77, 83, 133, 80)
        g.DrawLine(Pens.White, 77, 83, 133, 90)

        g.DrawLine(Pens.White, 77, 83, 10, 70)
        g.DrawLine(Pens.White, 77, 83, 13, 80)
        g.DrawLine(Pens.White, 77, 83, 13, 90)

        ' Рисуем глаза
        g.FillEllipse(Brushes.White, LX - 3, BY - 3, EyeRad, EyeRad)
        g.FillEllipse(Brushes.White, RX - 3, BY - 3, EyeRad, EyeRad)

        ' зрачки
        Dim LPupul As Point
        LPupul.X = CInt(LX - MoveRad * LeftCos)
        LPupul.Y = CInt(BY - MoveRad * LeftSin)
        g.FillEllipse(Brushes.Green, _
             LPupul.X, LPupul.Y, PupilRad, PupilRad)

        Dim RPupul As Point
        RPupul.X = CInt(RX - MoveRad * RightCos)
        RPupul.Y = CInt(BY - MoveRad * RightSin)
        g.FillEllipse(Brushes.Green, _
              RPupul.X, RPupul.Y, PupilRad, PupilRad)

        g.Dispose()
    End Sub
```

Данная программа основана на знании тригонометрических функций. Сначала вычисляется расстояние между зрачками и указателем мыши по известной теореме Пифагора. Затем, используя стороны воображаемого прямоугольного треугольника, находим синус и косинус угла между указателем мыши и его проекцией на координатные оси. Так как эти вычисления происходят динамически в событии таймера, то зрачки постоянно

знают, где в данный момент находится мышь, и поворачиваются на заданный угол (рис. 13.3).

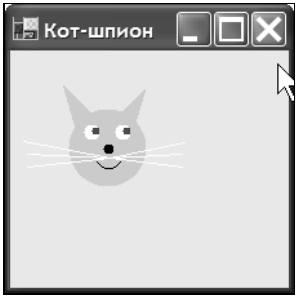

Рис. 13.3. Кот следит за мышиным указателем

13.14. Падал прошлогодний снег

А завершим мы нашу главу очень интересным примером, который приводит в восторг многих пользователей. Данная программа может послужить образцом для создания интерактивной поздравительной открытки или хранителя экрана. Запустив программу, вы увидите падающий снег, который оседает на различных предметах (рис. 13.4). Впервые, я увидел эту программу на сайте **http://www.emu8086.com/vb**. Тогда она была написана еще на Visual Basic 6.0. Но автор программы не стал бросать свое творение и переписал пример также для VB. NET 2003 и Java. В своих комментариях автор признался, что не знает, как сделать программу более гибкой. Дело в том, что в примере используется некоторая заранее подготовленная картинка. Я изменил программу таким образом, что можно не только использовать имеющуюся картинку, но и добавлять свои графические объекты (строки, прямоугольники, эллипсы). Проблема решалась очень просто. Нужно было получить объект `Graphics` не через метод `CreateGraphics`, а через функцию `FromImage`. Вы можете скачать с указанного сайта оригинал программы, который послужил прообразом моего примера, и сравнить полученные результаты (листинг 13.21).

Листинг 13.21. Падающий снег

```
'Copyright (C) 2004 Free Code
'http://www.emu8086.com/vb/
'Переделано А.Климовым 1 января 2005 года
```

```
'--------------------------------------
' Падающий снег © 2004 Александр Климов
'--------------------------------------
Public Class Form1
    Inherits System.Windows.Forms.Form

#Region " Windows Form Designer generated code "

    Public Sub New()
        MyBase.New()

        'This call is required by the Windows Form Designer.
        InitializeComponent()

        'Add any initialization after the InitializeComponent() call

    End Sub

    'Form overrides dispose to clean up the component list.
    Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
        If disposing Then
            If Not (components Is Nothing) Then
                components.Dispose()
            End If
        End If
        MyBase.Dispose(disposing)
    End Sub

    'Required by the Windows Form Designer
    Private components As System.ComponentModel.IContainer

    'NOTE: The following procedure is required by the Windows Form
     Designer
    'It can be modified using the Windows Form Designer.
    'Do not modify it using the code editor.
    Public WithEvents Timer1 As System.Windows.Forms.Timer
    <System.Diagnostics.DebuggerStepThrough()> Private Sub
     InitializeComponent()
        Me.components = New System.ComponentModel.Container
```

```
        Me.Timer1 = New System.Windows.Forms.Timer(Me.components)
        '
        'Timer1
        '
        Me.Timer1.Interval = 20
        '
        'Form1
        '
        Me.AutoScaleBaseSize = New System.Drawing.Size(6, 15)
        Me.ClientSize = New System.Drawing.Size(640, 260)
        Me.Name = "Form1"
        Me.Text = "Падал прошлогодний снег"

    End Sub

#End Region

    Structure xParticle
        Dim X As Integer
        Dim Y As Integer
        Dim oldX As Integer
        Dim oldY As Integer
        Dim iStopped As Integer
    End Structure

    Public Const MAXP As Integer = 400

    Public Snow(MAXP + 1) As xParticle


    Dim fMouseDown_X As Single
    Dim fMouseDown_Y As Single
    Dim bMOUSE_DOWN As Boolean

    Dim img As Image = _
          Image.FromFile(Application.StartupPath & "..\..\source.bmp")

    Dim g As Graphics
    Dim gr As Graphics = Graphics.FromImage(img)
```

```
    Dim b As Bitmap
    Dim TheBlackPixel As Color
    Dim mWidth As Integer
    Dim mHeight As Integer

    Private Sub Form1_Load(ByVal eventSender As System.Object, ByVal
eventArgs As System.EventArgs) Handles MyBase.Load
        InitSnow()
    End Sub

    Sub InitSnow()
        Randomize()
        g = Me.CreateGraphics
        Dim i As Integer

        b = New Bitmap(Me.ClientRectangle.Width + 1, _
           Me.ClientRectangle.Height + 1)
        b = b.FromFile(Application.StartupPath & "..\..\source.bmp")

        Dim gr As Graphics = Graphics.FromImage(b)

        ' Добавляем свою строку
        gr.DrawString("Visual Basic .NET", New Font("tahoma", 24),
Brushes.HotPink, 100, 50)

        ' For some reason "Color.Black" cannot be used, so this used:
        TheBlackPixel = b.GetPixel(0, 0)

        mWidth = b.Width - 1
        mHeight = b.Height - 1

        ' Draw first layer of snow for better look and
        ' snow falling algorithm calculations:
        For i = 0 To mWidth
            b.SetPixel(i, mHeight, Color.White)
        Next i

        ' The RND formula!
        ' Int((upperbound - lowerbound + 1) * Rnd + lowerbound)
```

```
        For i = 0 To MAXP
            newParticle(i)
            Snow(i).X = Int(mWidth * Rnd())
            Snow(i).Y = Int(mHeight * Rnd())
        Next i

        ' Устанавливаем размеры формы
        Dim hDif As Integer
        hDif = Me.Width - Me.ClientRectangle.Width
        Me.Width = mWidth + hDif + 1
        hDif = Me.Height - Me.ClientRectangle.Height
        Me.Height = mHeight + hDif + 1

        ' Включаем таймер
        Timer1.Enabled = True

    End Sub

    Sub DrawSnow()
        Dim i As Integer

        Dim newX As Integer
        Dim newY As Integer

        For i = 0 To MAXP
            b.SetPixel(Snow(i).oldX, Snow(i).oldY, TheBlackPixel)
            b.SetPixel(Snow(i).X, Snow(i).Y, System.Drawing.Color.White)
        Next i

        g.DrawImage(b, 0, 0)

        For i = 0 To MAXP
            Snow(i).oldX = Snow(i).X
            Snow(i).oldY = Snow(i).Y

            ' A trick to get both positive and negative random values:
            newX = Snow(i).X + Int(2 * Rnd())
            newX = newX - Int(2 * Rnd())
```

```
' Don't alow our snow to run away:
If newX < 0 Then newX = 0
If newX >= mWidth Then newX = mWidth - 1

If Snow(i).Y >= mHeight Then

    newParticle(i)

Else

    newY = Snow(i).Y + 1

    If b.GetPixel(newX, newY).Equals(TheBlackPixel) Then
        Snow(i).X = newX
        Snow(i).Y = newY
    Else
        '---------------------------
        If Snow(i).iStopped = 10 Then ' if stopped 10 times,
        make new!

            If Snow(i).X >= mWidth Then
                newParticle(i)
            ElseIf Snow(i).Y >= mHeight Then
                newParticle(i)
            Else
                ' Move according to basic SNOW RULE:
                If b.GetPixel(Snow(i).X + 1, Snow(i).Y +
                1).Equals(TheBlackPixel) Then
                    Snow(i).X = Snow(i).X + 1
                    Snow(i).Y = Snow(i).Y + 1
                    Snow(i).iStopped = 0
                ElseIf Snow(i).X > 0 Then
                    If b.GetPixel(Snow(i).X - 1, Snow(i).Y +
                    1).Equals(TheBlackPixel) Then
                        Snow(i).X = Snow(i).X - 1
                        Snow(i).Y = Snow(i).Y + 1
                        Snow(i).iStopped = 0
                    Else
                        newParticle(i)
```

```
                            End If
                        Else
                            newParticle(i)
                        End If

                    End If

                Else
                    Snow(i).iStopped = Snow(i).iStopped + 1
                End If

            End If

        End If

    Next i

End Sub

Sub newParticle(ByRef i As Integer)
    Snow(i).X = Int(mWidth * Rnd())
    Snow(i).Y = 0
    Snow(i).oldX = 0
    Snow(i).oldY = 0
    Snow(i).iStopped = 0
End Sub

    Private Sub Form1_Closing(ByVal eventSender As System.Object, ByVal
eventArgs As System.ComponentModel.CancelEventArgs) Handles My-
Base.Closing
        Dim Cancel As Integer = eventArgs.Cancel
        Timer1.Enabled = False
        g.Dispose()
        eventArgs.Cancel = Cancel
    End Sub

    Private Sub Timer1_Tick(ByVal eventSender As System.Object, ByVal
eventArgs As System.EventArgs) Handles Timer1.Tick
        DrawSnow()
    End Sub
```

```
    Private Sub Form1_Click(ByVal sender As Object, ByVal e As
System.EventArgs) Handles MyBase.Click
        InitSnow()  ' Начинаем сначала
    End Sub
End Class
```

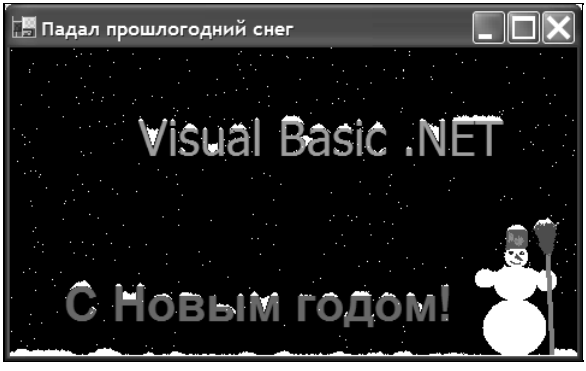

Рис. 13.4. Падающий снег

13.15. Советы и рекомендации

- В *разд. 13.3* курсор мыши движется в жестко заданной области 800×600 пикселов. Вам нужно написать универсальный пример, учитывающий реальные размеры монитора.
- Попробуйте оптимизировать программу *Убегающая кнопка*. Перепишите пример с использованием класса `Random` вместо функции `Rnd()`.
- Попробуйте написать пример, когда кнопка бегает за указателем мыши, уговаривая пользователя щелкнуть на ней.
- Попробуйте написать пример, скрывающий саму панель задач с экрана. Как найти дескриптор этого окна вы уже знаете.

Глава 14

Иллюзии

Иллюзия — это обман, базирующийся на человеческих слабостях. Самая типичная слабость — стереотипы. Иллюзия заставляет срабатывать стереотипы, вызывая смех или удовольствие автора, создавшего иллюзию. Но иногда иллюзии находят широкое применение во многих отраслях и сферах деятельности человека, в том числе и в программировании. Наиболее наглядный пример — иллюзия объемности кнопок и других элементов управления. В самом деле, ведь мониторы у нас плоские — однако мы видим, что кнопка на форме выпуклая, а при нажатии на нее — утапливается.

14.1. Иллюзия Орбисона

Если нарисовать серию концентрических окружностей и поместить на них квадрат, то квадрат будет казаться искаженным. Я заметил, что эффект усиливается, если квадрат помещать не в центр группы окружностей, а чуть-чуть сместить в сторону — проект `Orbison` (листинг 14.1).

Листинг 14.1. Иллюзия Орбисона

```
'---------------------------------------
' Иллюзии © 2004 Александр Климов
'---------------------------------------

    Private Sub mnuOrbison_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles mnuOrbison.Click
        Dim g As Graphics = CreateGraphics()
        g.Clear(BackColor)
        Dim redpen As Pen = Pens.Red
        g.DrawEllipse(redpen, 70, 70, 160, 160)
        g.DrawEllipse(redpen, 80, 80, 140, 140)
        g.DrawEllipse(redpen, 90, 90, 120, 120)
        g.DrawEllipse(redpen, 100, 100, 100, 100)
        g.DrawEllipse(redpen, 110, 110, 80, 80)
```

```
        g.DrawEllipse(redpen, 120, 120, 60, 60)
        g.DrawEllipse(redpen, 130, 130, 40, 40)
        g.DrawEllipse(redpen, 140, 140, 20, 20)
        g.DrawRectangle(redpen, 100, 100, 60, 60)
        g.Dispose()
    End Sub
```

Если вы не верите своим глазам (рис. 14.1) и думаете, что это у вас монитор передает фигуры с искажениями, то переделайте программу самостоятельно. Выведите сначала на экран квадрат. Убедитесь, что он выглядит правильно, а затем нарисуйте серию окружностей. Вы увидите, что монитор у вас правильный, просто срабатывает обман зрения.

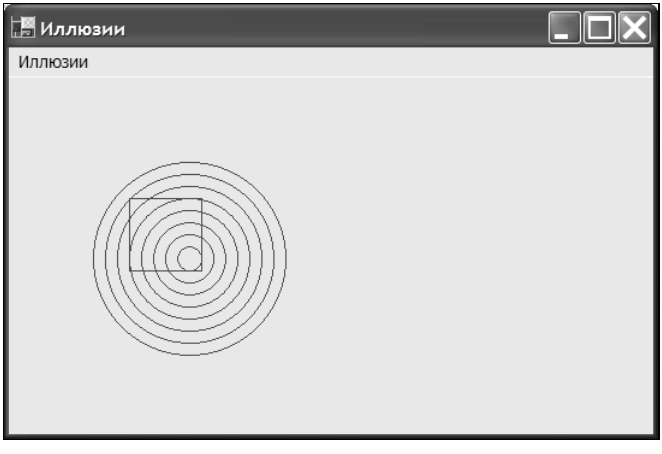

Рис. 14.1. Иллюзия Орбисона

14.2. Стена кафе (Wall Cafe)

Название этой иллюзии произошло от существовавшего в XIX веке кафе в городе Бристоль (Великобритания). Одна из стен кафе была выложена определенным образом чередующимися белыми и черными плитками. Воспроизведем данный эффект (листинг 14.2).

Листинг 14.2. Иллюзия стены кафе

```
    Private Sub mnuWallCafe_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles mnuWallCafe.Click
```

```
    Dim g As Graphics = CreateGraphics()
    g.Clear(BackColor)
    Dim blbr As SolidBrush = New SolidBrush(Color.Black)

    Dim counter As Integer
    For counter = 0 To 8
        g.FillRectangle(blbr, counter * 60, 10, 30, 30)
        g.DrawLine(New Pen(Color.Gray), 0, 40, 540, 40)
        g.FillRectangle(blbr, counter * 60 + 10, 40, 30, 30)
        g.DrawLine(New Pen(Color.Gray), 0, 70, 540, 70)
        g.FillRectangle(blbr, counter * 60 + 20, 70, 30, 30)
        g.DrawLine(New Pen(Color.Gray), 0, 100, 540, 100)
        g.FillRectangle(blbr, counter * 60 + 10, 100, 30, 30)
        g.DrawLine(New Pen(Color.Gray), 0, 130, 540, 130)
        g.FillRectangle(blbr, counter * 60, 130, 30, 30)
    Next
    g.Dispose()
End Sub
```

Для усиления иллюзии "стена кафе" разделена тонкими линиями (швами между плитками), которые позволяют получить более глубокий эффект путем разделения освещенностей между темными и светлыми элементами (рис. 14.2).

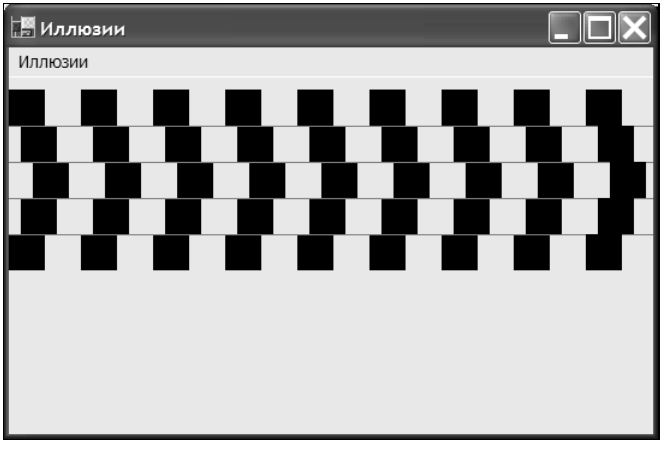

Рис. 14.2. Иллюзия с плитками

Из программного кода видно, что все прямоугольники одинаковы по размерам. А каковы ощущения? Давайте рассмотрим, отчего такое происходит. Дело в том, что светлые объекты на темном фоне видятся бóльшими, а темные — меньшими. И в данном случае размеры плиток мы рассматриваем относительно средней линии. Поэтому, если в нижней полосе идет чередование черного и белого цветов, в то время, как на верхней находится белый блок, то средняя линия будет "подниматься". Если наоборот, то опускаться. Если вы задумали сделать ремонт в ванной и пригласили плиточника выложить плитку подобным образом, то не спешите его ругать за плохо сделанную работу. Поверьте, плиточник здесь не причем. Зато у вас будет самая оригинальная облицовка, которая станет достопримечательностью среди ваших знакомых.

14.3. Параллельны ли прямые?

Из курса школьной математики мы знаем, что параллельные прямые не пересекаются. Но так ли это? Давайте попробуем нарисовать две параллели и сделать на них зарубки (листинг 14.3).

Листинг 14.3. Параллельные прямые

```
Dim g As Graphics = CreateGraphics()
g.Clear(Color.Black)

' Первый эффект
g.DrawLine(New Pen(Color.White), 50, 50, 500, 50)
g.DrawLine(New Pen(Color.White), 50, 70, 500, 70)

Dim i As Integer
For i = 50 To 480 Step 10
    g.DrawLine(New Pen(Color.White), i, 45, i + 10, 55)
    g.DrawLine(New Pen(Color.White), i, 75, i + 10, 65)
Next
g.Dispose()
```

Запустив проект, вы увидите, что прямые не совсем параллельны, а расходятся в правом углу картинки (рис. 14.3, вверху). Выходит, что учителя нас все эти годы обманывали? Не пора ли пересмотреть учебники математики? Вот еще одно доказательство непараллельности параллельных прямых (листинг 14.4).

Листинг 14.4. Второй эффект с параллельными прямыми

```
' Второй эффект
Dim g As Graphics = CreateGraphics()
g.Clear(Color.Black)

g.DrawLine(New Pen(Color.White), 10, 100, 450, 100)
g.DrawLine(New Pen(Color.White), 10, 180, 450, 180)

Dim i As Integer
For i = -30 To 430 Step 30
    g.DrawLine(New Pen(Color.White), 230, 150, i + 30, 80)
    g.DrawLine(New Pen(Color.White), 230, 150, i + 30, 190)
Next
g.Dispose()
```

Теперь вы видите, что по краям параллельные прямые имеют свойство сужаться (рис. 14.3, внизу). И если мысленно дополнить эти прямые, то несомненно они скоро пересекутся. Если читатель еще учится в школе, то покажите этот пример вашему учителю математики. За возможные последствия автор книги ответственности не несет! Используйте данные примеры на собственный страх и риск.

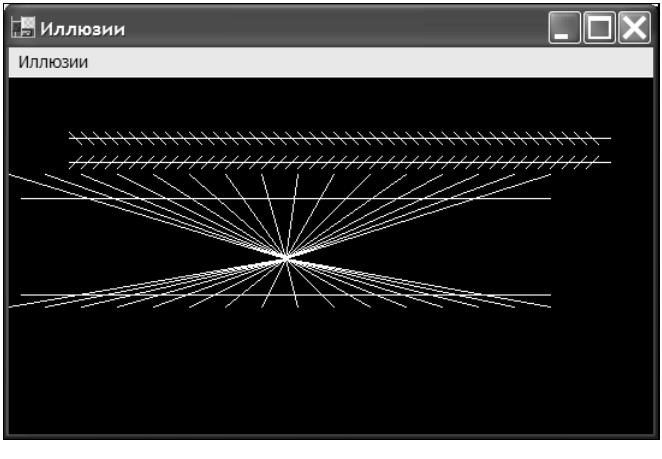

Рис. 14.3. Пересекутся ли параллельные прямые?

14.4. Объемные фигуры

Иллюзии, кроме своей занимательности, имеют и практическую ценность. В *главе 2* мы рассматривали вопрос создания объемного текста. Иллюзия создания объемности широко используется в графических операционных системах. Когда на компьютерах большинства пользователей стояла операционная система DOS, выход Windows 95 с графическим интерфейсом ошеломил миллионы людей. Всем понравился внешний вид приложений с объемными элементами управления.

Мы попробуем пойти по стопам разработчиков и придать объемность своим фигурам. В качестве примера возьмем такую простую фигуру, как прямоугольник. Поняв принцип создания объемности, вы без труда сможете создавать собственные объемные кнопки, надписи и т. д. Если внимательно присмотреться к фигуре, которая кажется нам объемной, то можно заметить, что подобный эффект достигается использованием тонких линий разного цвета по краям прямоугольника. Давайте сначала разместим на форме надпись `Label1` как образец для подражания. Присвойте ее свойству `BorderStyle` значение `Fixed3D` и удалите текст из свойства `Text`. Размеры и местоположения не имеют особого значения. Я присвоил им следующие значения:

- `Location = 10;10;`
- `Size = 50;40.`

Запустите проект и тщательно рассмотрите надпись. Как видите, левая и верхняя части надписи обрамлены линиями темно-серого цвета, а нижняя и правая — белого цвета. Запомним эту информацию. Для удобства мы поместим код рисования аналогичного эффекта в отдельную процедуру (листинг 14.5).

Листинг 14.5. Создание иллюзии выпуклости

```
'--------------------------------------
' Объемные фигуры © 2004 Александр Климов
'--------------------------------------
    Private Sub Draw3DBorder(ByVal g As Graphics, ByVal rect As Rectan-
gle, Optional ByVal sunken As Boolean = True)
        ' Процедура придания выпуклости

        Dim colors() As Color

        If sunken Then
            colors = New Color() { _
```

```
                SystemColors.ControlDark, _
                SystemColors.ControlLightLight _
            }
        Else
            colors = New Color() { _
                SystemColors.ControlLightLight, _
                SystemColors.ControlDark _
            }
        End If

        Dim p As Pen
        p = New Pen(colors(0))
        g.DrawLine(p, rect.X, rect.Y + rect.Height - 1, rect.X, rect.Y)
        g.DrawLine(p, rect.X, rect.Y, rect.X + rect.Width - 1, rect.Y)

        p = New Pen(colors(1))
        g.DrawLine(p, rect.X, rect.Y + rect.Height - 1, _
           rect.X + rect.Width - 1, rect.Y + rect.Height - 1)
        g.DrawLine(p, rect.X + rect.Width - 1, _
           rect.Y + rect.Height - 1, rect.X + rect.Width - 1, rect.Y)

    End Sub
```

Разберем подробнее, что делает эта процедура. Сначала я объявил массив, в котором будут содержаться цвета линий, создающих эффект объемности. Далее идет условие `If ... Else`. Мы пока пропустим второе условие, а рассмотрим условие под названием `sunken`. Этот кусочек кода реализует режим вдавливания. Для этого нам понадобятся два цвета, специально разработанных Microsoft для данного режима: `ControlDark` и `ControlLightLight`. Более подробно об этих цветах вы можете почитать в документации. Для рисования прямоугольника возьмем не метод `DrawRectangle`, а четыре вызова метода `DrawLine`. Причем первые два метода рисуют цветом `ControlDark`, а следующие используют цвет `ControlLightLight`. Если вы помните, я просил вас обратить внимание, как выглядит надпись `Label1`. Теперь у нас есть возможность визуально сравнить надпись `Label1` с нашей рукотворной фигурой. Для этого в обработчик события `Form1_Paint` введем следующий код:

```
Draw3DBorder(e.Graphics, _
    New Rectangle(70, 10, _
        50, 40), _
    True)
```

Так как я расположил надпись в координатах (10, 10) с шириной в 50 пикселов, то свою фигуру для сравнения я разместил на 10 пикселов правее `Label1`. Надеюсь, вы согласитесь, что созданная программно фигура ничем по внешнему виду не отличается от надписи. Наверное, вы уже догадались, что вторая часть условия `If ... Else` позволяет рисовать выпуклую фигуру:

```
Else
     colors = New Color() { _
        SystemColors.ControlLightLight, _
        SystemColors.ControlDark _
     }
End If
```

Причем, для достижения эффекта выпуклости требуется всего лишь поменять местами прежние цвета. Чтобы выяснить, как будет выглядеть выпуклая фигура на экране, добавим в событие `Form1_Paint` код:

```
Draw3DBorder(e.Graphics, _
            New Rectangle(70, 60, _
            50, 40), _
            False)
```

В результате прямо под надписью `Label1` у нас появится фигура, напоминающая кнопку. Но это еще не все. У Microsoft в запасе осталось еще два цвета, которые служат для придания фигурам объемного вида: `ControlDarkDark` и `ControlLight`. Применение этих цветов наряду с теми двумя, о которых рассказывалось ранее, дает более вдавленную или выпуклую картинку. Для этой цели я поместил использование всех четырех цветов в новую процедуру `Draw3DBorder2` (листинг 14.6).

Листинг 14.6. Вторая процедура для создания объемности фигуры

```
    Private Sub Draw3DBorder2(ByVal g As Graphics, ByVal rect As
Rectangle, Optional ByVal sunken As Boolean = True)
        ' Процедура придания более глубокой вдавленности

        Dim colors() As Color

        If sunken Then
            colors = New Color() { _
                SystemColors.ControlDark, _
                SystemColors.ControlDarkDark, _
                SystemColors.ControlLightLight, _
```

```
            SystemColors.ControlLight _
        }
    Else
        colors = New Color() { _
            SystemColors.ControlLightLight, _
            SystemColors.ControlLight, _
            SystemColors.ControlDark, _
            SystemColors.ControlDarkDark _
        }
    End If

    Dim p As Pen
    p = New Pen(colors(0))
    g.DrawLine(p, rect.X, rect.Y + rect.Height - 1, rect.X, rect.Y)
    g.DrawLine(p, rect.X, rect.Y, rect.X + rect.Width - 1, rect.Y)
    p = New Pen(colors(1))
    g.DrawLine(p, rect.X + 1, rect.Y + rect.Height - 2, rect.X + 1,
    rect.Y + 1)
    g.DrawLine(p, rect.X + 1, rect.Y + 1, rect.X + rect.Width - 2,
    rect.Y + 1)
    p = New Pen(colors(2))
    g.DrawLine(p, rect.X, rect.Y + rect.Height - 1, rect.X +
    rect.Width - 1, rect.Y + rect.Height - 1)
    g.DrawLine(p, rect.X + rect.Width - 1, rect.Y + rect.Height - 1,
    rect.X + rect.Width - 1, rect.Y)
    p = New Pen(colors(3))
    g.DrawLine(p, rect.X + 1, rect.Y + rect.Height - 2, rect.X +
    rect.Width - 2, rect.Y + rect.Height - 2)
    g.DrawLine(p, rect.X + rect.Width - 2, rect.Y + rect.Height - 2,
    rect.X + rect.Width - 2, rect.Y + 1)

End Sub
```

Ничего принципиально нового в этой процедуре нет, поэтому пояснений по коду я давать не буду. Разберите пример самостоятельно. Вот как будет выглядеть вызов двух созданных процедур в событии `Form1_Paint` (листинг 14.7).

Листинг 14.7. Вызов процедур в программе

```
    Private Sub Form1_Paint(ByVal sender As Object, ByVal e As
System.Windows.Forms.PaintEventArgs) Handles MyBase.Paint
```

```
        Draw3DBorder(e.Graphics, _
            New Rectangle(70, 10, _
                50, 40), _
            True)

        Draw3DBorder(e.Graphics, _
                    New Rectangle(10, 60, _
                                  50, 40), _
                    False)

        Draw3DBorder2(e.Graphics, _
                    New Rectangle(10, 110, _
                                  50, 40), _
                    True)

        Draw3DBorder2(e.Graphics, _
                    New Rectangle(70, 110, _
                                  50, 40), _
                    False)

    End Sub
```

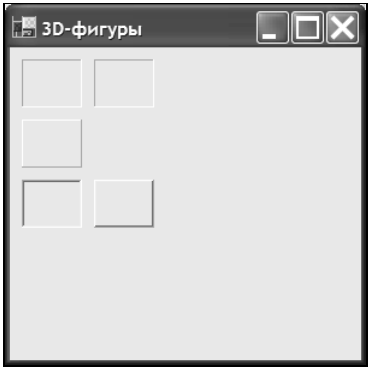

Рис. 14.4. Создание объемных фигур

В ваших руках мощная заготовка, которую можно использовать в своих целях. Все зависит от вашей фантазии. Теперь, когда вы знаете, как програм-

мисты из Microsoft создавали стандартные элементы управления, то можете смело заявлять своим малоопытным друзьям, что в ваши руки попали исходники Windows.

14.5. Советы и рекомендации

Я привел всего несколько простейших примеров иллюзий. На самом деле их гораздо больше. Очень часто картинки, в которых используется обман зрения, публикуются в книгах и журналах, посвященных головоломкам. Попробуйте сами воспроизвести подобные трюки на своем компьютере. Возможно, часть таких иллюзий найдет свое применение при разработке собственных приложений.

Глава 15

Звуки музыки

Эта глава посвящена воспроизведению звуков. Современный компьютер уже невозможно представить себе без устройств мультимедиа, позволяющих слушать музыку, играть в игры, создавать собственные мелодии.

15.1. Воспроизведение звукового файла

Возьмем самый простой случай. Попытаемся воспроизвести звуковой файл формата WAV. На данный момент .NET Framework не имеет встроенных средств для воспроизведения звуковых файлов, поэтому воспользуемся средствами Windows API. Объявим в начале нового проекта `PlaySound` одноименную функцию `PlaySound` и несколько сопутствующих констант. Чтобы воспроизвести звуковой файл, достаточно вызвать объявленную функцию с указанным именем файла (листинг 15.1).

Листинг 15.1. Воспроизведение звукового файла

```
'--------------------------------------
' Воспроизведение звука © 2004 Александр Климов
'--------------------------------------
    Public Const SND_SYNC = &H0
    Public Const SND_ASYNC = &H1
    Public Const SND_FILENAME = &H20000
    Public Const SND_RESOURCE = &H40004
    Private Declare Auto Function PlaySound Lib "winmm.dll" _
           (ByVal name As String, _
            ByVal hmod As Integer, ByVal flags As Integer) As Integer


   Private Sub butPlay_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles butPlay.Click
```

```
        PlaySound(Application.StartupPath & "../../meow.wav", _
                              Nothing, SND_FILENAME Or SND_ASYNC)
    End Sub
```

В этом примере программа ищет указанный звуковой файл и воспроизводит его. Обращаю ваше внимание, что функция воспроизводит только файлы с расширением wav. К сожалению, столь популярный формат, как MP3, функцией `PlaySound` не поддерживается.

Примечание

В новой версии Visual Basic .NET 2005 разработчики обещают добавить новый класс `SoundPlayer`, который позволит воспроизводить WAV-файлы без использования неуправляемого кода Windows API.

15.2. А есть ли звуковая карта?

Рассматривая пример с проигрыванием звукового файла, мы упустили один момент. А может ли вообще компьютер пользователя проигрывать звуковые файлы? Ситуация вполне обычная для офисных компьютеров. В этом случае неплохо бы проверить наличие на компьютере устройств, способных воспроизводить звуки. Для этого можно воспользоваться функцией Windows API `waveOutGetNumDevs`:

```
Private Declare Function waveOutGetNumDevs Lib "winmm" () As Integer
```

Данная функция в успешном случае возвращает число устройств в системе, способных работать со звуком. Таким образом, любое число, отличное от нуля, свидетельствует о наличии звуковой карты на компьютере пользователя. Можно оформить вызов данной функции в виде собственной функции и в каком-нибудь месте кода вызвать эту функцию (листинг 15.2).

Листинг 15.2. Проверка на наличие звуковой карты

```
    Function IsSoundCardPresent() As Boolean
        Return waveOutGetNumDevs() >= 1
    End Function
    Private Sub butSoundCard_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles butSoundCard.Click
        MsgBox(IsSoundCardPresent)
    End Sub
```

Анекдот в тему

Замученный программист, у которого постоянно глючит компьютер, звонит на радио в программу по заявкам.

– Поставьте для меня песню Аллы Пугачевой "Кликну, а в ответ тишина".

15.3. Устройства для записи звука

Кроме проигрывания звуковых файлов, компьютер может и записывать звук с микрофона. Чтобы убедиться, что компьютер обладает такой возможностью, можно использовать похожую функцию из предыдущего примера `waveInGetNumDevs`:

```
Private Declare Function waveInGetNumDevs Lib "winmm.dll" () As Integer
```

Теперь достаточно вызвать эту функцию и получить число устройств, способных записывать звук (листинг 15.3).

Листинг 15.3. Получение числа устройств для записи звука

```
    Private Sub butIn_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles butIn.Click
        lblInfo.Text = "Доступно " & waveInGetNumDevs() & " устройства
        записи"
    End Sub
```

На моем компьютере оказалось целых два таких устройства!

15.4. Воспроизведение MIDI-файлов

Итак, функция `PlaySound` может воспроизводить только звуковые файлы в формате WAV. Но существует еще один распространенный музыкальный формат MIDI. Как быть с такими файлами? Придется использовать еще одну функцию Windows API `mciSendString`. Мы уже использовали эту функцию для открывания и закрывания привода CD-ROM (*см. главу 13*), но возможности этой функции гораздо шире. Именно она поможет нам воспроизвести файл MIDI. В состав Windows входят несколько таких файлов, я выбрал один из них (листинг 15.4).

Листинг 15.4. Воспроизведение MIDI

```
    ' Функция для воспроизведения midi
    Private Declare Function mciSendString Lib "winmm.dll" _
```

```
                                        Alias "mciSendStringA" _
                                       (ByVal lpstrCommand As String, _
                                        ByVal lpstrReturnString As String, _
                                        ByVal uReturnLength As Long, _
                                        ByVal hwndCallback As Long) As Long

    Public Sub butMidi_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles butMidi.Click
        Dim lRet As Long

        Dim testmidi As String = "C:\Windows\Media\town.mid"

        ' Открываем устройство для проигрывания файла MIDI
        lRet = mciSendString( _
          "open " & testmidi & " type sequencer alias mplayer", _
           0&, 0, 0)

        ' Начинаем воспроизведение
        lRet = mciSendString("play mplayer", 0&, 0, 0)

    End Sub

    Private Sub butCloseMidi_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles butCloseMidi.Click
        ' Закрываем файл и устройство
        mciSendString("close mplayer", 0&, 0, 0)
    End Sub
```

Надо быть осторожными при воспроизведении файлов MIDI в своих программах. Всегда нужно соблюдать определенную последовательность действий: открыть файл и устройство, начать проигрывать, остановить устройство, закрыть устройство и файл. Невыполнение этих правил в некоторых случаях может привести к зависанию вашей программы. Более подробно о работе с файлами MIDI можно почитать на сайте **http://msdn.microsoft.com**.

15.5. Виртуальное пианино

Если мы можем программно проигрывать музыкальный файл, значит, у нас есть возможность создать свой виртуальный музыкальный инструмент, например, пианино. У пианино, как и у клавиатуры, есть клавиши. Восполь-

зуемся этим обстоятельством. Идея состоит в следующем. Нарисуем на форме ряд клавиш, соответствующих расположению настоящих клавиш на пианино. Сопоставим каждой клавише музыкальный звук. Создадим в программе клавиатурный интерфейс. После выполнения этих задач можно запустить программу и стучать по определенным клавишам клавиатуры, извлекая нужную мелодию. Участие в телепроекте "Фабрика звезд" я вам не обещаю, но порадовать своих близких вполне возможно.

Я создал предварительную версию программы. Ваша задача — доработать ее под свои нужды. Проект `VBPiano` начинается с того, что сначала надо найти звуковые файлы, которые играют определенные ноты. На прилагаемом компакт-диске вы найдете несколько таких файлов из моей коллекции. Затем надо попытаться придать кнопкам некоторую похожесть на клавиши пианино. Кстати, чтобы избежать нудной работы по размещению каждой кнопки на форме, можно воспользоваться динамическим созданием массива кнопок с заданными свойствами. Я пошел старым путем и вручную разместил на форме 20 кнопок. Двенадцать из них должны быть окрашены в белый цвет, остальные — в черный. Для большей наглядности я разместил также 12 надписей `Label`, которые будут исполнять роль цветовых индикаторов (листинг 15.5).

Листинг 15.5. Создание виртуального пианино

```
'---------------------------------------
' Виртуальное пианино © 2004 Александр Климов
'---------------------------------------
    Public Const SND_ASYNC = &H1
    Public Const SND_NODEFAULT = &H2
    Public Const SND_FILENAME = &H20000

    Private Declare Auto Function PlaySound Lib "winmm.dll" _
              (ByVal name As String, ByVal hmod As Integer, _
               ByVal flags As Integer) As Integer

    Private Sub Form1_KeyDown(ByVal sender As Object, ByVal e As
System.Windows.Forms.KeyEventArgs) Handles MyBase.KeyDown

        Select Case e.KeyCode
            Case Keys.D1
                Button1.PerformClick()
            Case Keys.D2
                Button2.PerformClick()
```

```
        Case Keys.D3
            Button3.PerformClick()
        Case Keys.D4
            Button4.PerformClick()
        Case Keys.D5
            Button5.PerformClick()
        Case Keys.D6
            Button6.PerformClick()
        Case Keys.D7
            Button7.PerformClick()
        Case Keys.D8
            Button8.PerformClick()
        Case Keys.D9
            Button9.PerformClick()
        Case Keys.D0
            Button10.PerformClick()
        Case Keys.OemMinus
            Button11.PerformClick()
        Case Keys.Oemplus
            Button4.PerformClick()
        Case Keys.F1
            Button13.PerformClick()
        Case Keys.F2
            Button14.PerformClick()
        Case Keys.F3
            Button15.PerformClick()
        Case Keys.F4
            Button16.PerformClick()
        Case Keys.F5
            Button17.PerformClick()
        Case Keys.F6
            Button18.PerformClick()
        Case Keys.F7
            Button19.PerformClick()
        Case Keys.F8
            Button20.PerformClick()
    End Select
End Sub
```

```
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
        Dim filename As String
        Const wavefile As String = "..\..\1.wav"
        filename = Application.StartupPath & wavefile
        PlaySound(filename, Nothing, SND_FILENAME Or SND_ASYNC)
    End Sub

    Private Sub Button1_MouseUp(ByVal sender As Object, ByVal e As
System.Windows.Forms.MouseEventArgs) Handles Button1.MouseUp
        Label1.BackColor = Color.Chartreuse
    End Sub

    Private Sub Button1_MouseDown(ByVal sender As Object, ByVal e As
System.Windows.Forms.MouseEventArgs) Handles Button1.MouseDown
        Label1.BackColor = Color.Green
    End Sub
```

Сам код не представляет никакой сложности. Нажатие на кнопку извлекает звук из ваших колонок. Одновременно загорается соответствующий цветовой индикатор `Label` (рис. 15.1). Так как код для остальных кнопок ничем не отличается от кода первой кнопки, я из экономии места не привожу его в книге. Отдельно только можно остановиться на событии `Form_KeyDown`. У формы нужно установить свойство `KeyPreview` в `True`. В этом случае мы сможем обрабатывать не только щелчки мыши на кнопках, но и нажатия на клавиши. Для примера я использовал клавиши двух верхних рядов стандартной клавиатуры: 8 функциональных клавиш (<F1>, <F2>) и 12 цифровых клавиш (<1>, <2>, <3>). Запустите программу и начинайте нажимать на клавиши клавиатуры. Только пожалуйста, приглушите громкость своих динамиков!

Рис. 15.1. Виртуальное пианино

Анекдот в тему

Приходит программист к музыканту в гости. Музыкант хвалится свежеприобретенным пианино, программист оценивающе смотрит и выдает: "Клавиатура конечно хреновая, всего 89 кнопок, но то, что кнопку <Shift> ногами надо нажимать, это круто!"

15.6. Проигрыватель WinAmp

Одним из самых известных проигрывателей музыкальных композиций является программа WinAmp (**http://www.winamp.com**). В этом примере я хочу показать, как можно получить в своей программе название проигрываемой песни из WinAmp. Для этого нам понадобится помощь функций Windows API `FindWindow` и `GetWindowText`. С помощью данных функций мы можем найти окно программы WinAmp и получить текст ее заголовка. На сайте создателей проигрывателя можно узнать, что имя класса проигрывателя `Winamp v1.x`. Кроме того, проигрыватель всегда добавляет к названию проигрываемой композиции строчку " - `Winamp`". Зная имя класса, мы без труда можем получить дескриптор окна, а затем и заголовок окна. Данный код я оформил в виде отдельной функции `GetWinampTitle` (листинг 15.6).

Листинг 15.6. Получение названия песни из WinAmp

```
'---------------------------------------
' WinAmp © 2004 Александр Климов
'---------------------------------------
    Private Declare Auto Function FindWindow Lib "user32" ( _
      ByVal lpClassName As String, _
      ByVal lpWindowName As String) As IntPtr
    Private Declare Auto Function GetWindowText Lib "user32" ( _
       ByVal hwnd As IntPtr, _
       ByVal lpString As String, _
       ByVal cch As Integer) As Integer
     Private Const lpClassName = "Winamp v1.x"
     Private Const strTtlEnd = " - Winamp"

     Private Function GetWinampTitle() As String
        ' дескриптор окна Winamp
        Dim hwnd As IntPtr
        hwnd = FindWindow(lpClassName, vbNullString)
```

```
        ' заголовок окна
        Dim lpText As String

        If hwnd.Equals(IntPtr.Zero) Then Return "WinAmp не запущен!"

        lpText = New String(Chr(0), 100)
        Dim intLength As Integer = GetWindowText(hwnd, lpText, _
          lpText.Length)

        If (intLength <= 0) OrElse (intLength > lpText.Length) _
                 Then Return "Не опознано"

        Dim strTitle As String = lpText.Substring(0, intLength)
        Dim intName As Integer = strTitle.IndexOf(strTtlEnd)

        If (strTitle.EndsWith(strTtlEnd)) AndAlso _
            (strTitle.Length > strTtlEnd.Length) Then _
            strTitle = strTitle.Substring(0, _
              strTitle.Length - strTtlEnd.Length)

        Dim intDot As Integer = strTitle.IndexOf(".")
        If (intDot > 0) AndAlso (IsNumeric( _
               strTitle.Substring(0, intDot))) Then _
            strTitle = strTitle.Remove(0, intDot + 1)

        Return strTitle.Trim
    End Function

   Private Sub butGetSong_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles butGetSong.Click
        TextBox1.Text = GetWinampTitle()
    End Sub
```

Функция `GetWinampTitle` получилась немного длинной. Но подобный подход оправдан. Дело в том, что если бы мы ограничились просто вызовом функции `GetWindowText`, то получили бы избыточную информацию. Во-первых, в найденной строке будет содержаться номер песни из списка воспроизведения (playlist), за которым затем идут точка и пробел. Далее идет интересующее нас название песни, которое завершается строчкой " - Winamp". Таким образом, нам надо извлечь полезную информацию из длинной строки, что и делает

данная функция. Теперь, чтобы получить название текущей песни, достаточно просто вызвать функцию `GetWinampTitle` без всяких параметров. Не забудьте разместить на форме кнопку и текстовое поле, в котором будет отображаться название песни.

Примечание

Если сделать паузу или остановить песню, то в заголовке окна WinAmp появятся дополнительные слова в квадратных скобках **[Paused]** или **[Stopped]**. Стало быть, нужно предусмотреть и такую ситуацию.

15.7. Теги MP3-файлов

Рассмотренный только что пример предполагает наличие у пользователя программы WinAmp. А что делать, если пользователь не хочет устанавливать эту программу? Тогда можно предложить ему свой проигрыватель MP3-файлов! И здесь уместно подробнее рассказать о структуре файла MP3. Наверное, вы замечали, что при воспроизведении MP3-файлов проигрыватели показывают иногда дополнительную информацию о песне — ее название, автора, год выпуска, название альбома. У вас не возникало вопроса, откуда проигрыватель берет эту информацию? Оказывается, в файлах с расширением mp3 зарезервировано несколько байтов, в которые и записываются необходимые данные в специальном формате. Зная такой способ упаковки данных, проигрыватели извлекают нужную информацию.

Я раскрою перед вами этот секрет, и вы самостоятельно сможете прочитать скрытые данные. Итак, слушайте меня внимательно. Файлы MP3 могут хранить различную информацию об исполнителе, названии песни, альбома и т. д. Данная информация хранится в так называемом *теге* (tag) ID3. Располагается этот тег в последних 128 байтах MP3-файла (но его наличие в файле не обязательно). Таким образом, для извлечения информации нужно считать последние 128 байтов файла, проверить специальную сигнатуру (первые три символа — `TAG`), и если она совпала, то считаем эти самые 128 байтов ID3-тегом. Сам тег, в свою очередь, делится на небольшие фрагменты по несколько байтов, в которых и содержится информация. Данные о содержимом ID3-тега версии 1.0 приведены в табл. 15.1.

Таблица 15.1. *Содержимое тега ID3 версии 1.0*

Байт	Кол-во символов	Название	Содержание
0	Char[3]	Signature	"TAG" (если сигнатуры нет, то это не ID3-тег)
3	Char[30]	Title	Название песни

Таблица 15.1 *(окончание)*

Байт	Кол-во символов	Название	Содержание
33	Char[30]	Artist	Исполнитель
63	Char[30]	Album	Название альбом
93	Char[4]	Year	Год выпуска
97	Char[30]	Comment	Комментарий
127	byte	Genre	Жанр

Как видите, название песни, альбома, исполнитель и комментарий представляются не более чем 30 символами. Если же в строке меньше 30 символов, то она дополняется символами с кодом 0 (первый символ с кодом 0 считается концом строки). Имеется также небольшое расширение формата — ID3-тег версии 1.1, позволяющий сохранять номер трека компакт-диска, с которого оцифровывалсь музыка. Для этого выделен последний байт комментария. Чтобы быть уверенным, что это именно номер трека, а не символ из очень длинного комментария, надо проверить предпоследний байт на равенство нулю. В нашем примере мы будем пользоваться версией 1.1, так как она более распространена на данный момент. Кроме того, уже существует спецификация ID3 версии 2.0, в которую вошли дополнительные данные. Более подробную информацию о спецификации MP3-тегов вы можете без труда найти в Интернете. Однако вернемся к табл. 15.1. Последний байт тега отвечает за название жанра. Была разработана таблица соответствия названий жанров значению байта. Я приведу официальные сведения с использованием английского языка (табл. 15.2), а в примере мы попробуем переделать эту таблицу, чтобы она была пригодна для использования на великом и могучем русском языке.

Таблица 15.2. *Соответствие названий жанров значению последнего байта тега ID3 версии 2.0*

Жанр	Значение байта	Жанр	Значение байта
Blues	0	Grunge	6
ClassicRock	1	HipHop	7
Country	2	Jazz	8
Dance	3	Metal	9
Disco	4	NewAge	10
Funk	5	Oldies	11

Таблица 15.2 *(продолжение)*

Жанр	Значение байта	Жанр	Значение байта
Other	12	Soul	42
Pop	13	Punk	43
RnB	14	Space	44
Rap	15	Meditative	45
Reggae	16	InstrumentalPop	46
Rock	17	InstrumentalRock	47
Techno	18	Ethnic	48
Industrial	19	Gothic	49
Alternative	20	Darkwave	50
Ska	21	TechnoIndustrial	51
DeathMetal	22	Electronic	52
Pranks	23	PopFolk	53
Soundtrack	24	Eurodance	54
EuroTechno	25	Dream	55
Ambient	26	SouthernRock	56
TripHop	27	Comedy	57
Vocal	28	Cult	58
JazzFunk	29	Gangsta	59
Fusion	30	Top40	60
Trance	31	ChristianRap	61
Classical	32	PopFunk	62
Instrumental	33	Jungle	63
Acid	34	NativeAmerican	64
House	35	Cabaret	65
Game	36	NewWave	66
SoundClip	37	Psychadelic	67
Gospel	38	Rave	68
Noise	39	Showtunes	69
AlternRock	40	Trailer	70
Bass	41	LoFi	71

***Таблица 15.2** (окончание)*

Жанр	Значение байта	Жанр	Значение байта
Tribal	72	Musical	77
AcidPunk	73	RocknRoll	78
AcidJazz	74	HardRock	79
Polka	75	None	255
Retro	76		

Не исключено, что названия жанров будут пополняться. Поищите информацию также в Интернете.

После краткого введения в теорию пора приступать к практике. Очень хороший пример я однажды нашел на сайте **http://www.codeproject.com**. Этот пример и стал отправной точкой для моего проекта. Запустите Visual Basic .NET 2003 и создайте новый проект `MP3`. Для извлечения информации из MP3-файлов мы создадим отдельный класс `MP3TAG`. В проекте щелкните правой кнопкой мыши на названии проекта и выберите последовательно команды **Add | Add Class**. У вас откроется диалоговое окно создания нового класса. Измените имя `Class.vb`, предлагаемое по умолчанию, на `MP3TAG.vb`. Начнем с наполнения созданного класса. Для начала импортируем пространство имен `System.IO`, которое осуществляет чтение и запись файлов (листинг 15.7).

Листинг 15.7. Извлечение информации из файлов MP3

```
'--------------------------------------
' Теги MP3-файлов © 2004 Александр Климов
'--------------------------------------
' Класс для работы с тегами MP3
Imports System.IO
Public Class MP3TAG
    ' Конструктор
    Public Sub New(Optional ByVal Filename As String = "")
        MyBase.New()
        If (Filename <> "") Then Me.Filename = Filename
    End Sub

    ' Жанры
    Public Enum Genres As Byte
```

```
Blues = 0
ClassicRock = 1
Country = 2
Dance = 3
Disco = 4
Funk = 5
Grunge = 6
HipHop = 7
Jazz = 8
Metal = 9
NewAge = 10
Oldies = 11
Other = 12
Pop = 13
RnB = 14
Rap = 15
Reggae = 16
Rock = 17
Techno = 18
Industrial = 19
Alternative = 20
Ska = 21
DeathMetal = 22
Pranks = 23
Soundtrack = 24
EuroTechno = 25
Ambient = 26
TripHop = 27
Vocal = 28
JazzFunk = 29
Fusion = 30
Trance = 31
Classical = 32
Instrumental = 33
Acid = 34
House = 35
Game = 36
SoundClip = 37
Gospel = 38
```

```
Noise = 39
AlternRock = 40
Bass = 41
Soul = 42
Punk = 43
Space = 44
Meditative = 45
InstrumentalPop = 46
InstrumentalRock = 47
Ethnic = 48
Gothic = 49
Darkwave = 50
TechnoIndustrial = 51
Electronic = 52
PopFolk = 53
Eurodance = 54
Dream = 55
SouthernRock = 56
Comedy = 57
Cult = 58
Gangsta = 59
Top40 = 60
ChristianRap = 61
PopFunk = 62
Jungle = 63
NativeAmerican = 64
Cabaret = 65
NewWave = 66
Psychadelic = 67
Rave = 68
Showtunes = 69
Trailer = 70
LoFi = 71
Tribal = 72
AcidPunk = 73
AcidJazz = 74
Polka = 75
Retro = 76
Musical = 77
```

```
    RocknRoll = 78
    HardRock = 79
    None = 255
End Enum

' Информация о песне
Public Enum Tag_Info As Byte
    Title = 0   ' название песни
    Artist = 1  ' исполнитель
    Album = 2   ' название альбома
    Year = 3    ' год выпуска
    Track = 4   ' номер дорожки
    Comment = 5 ' комментарий
    Genre = 6   ' жанр
End Enum

' Имя файла
Private strFileName As String
Public Property Filename() As String
    Get
        Return strFileName
    End Get
    Set(ByVal Value As String)
        Dim checkFile As File
        If (checkFile.Exists(Value)) Then
            strFileName = Value
            Refresh()
        Else
            Throw New System.IO.FileLoadException( _
                "Указанный файл не существует", Value)
        End If
    End Set
End Property

' Есть ли тег в файле?
Private bTagExists As Boolean
Public ReadOnly Property TagExists() As Boolean
    Get
        Return bTagExists
```

```
    End Get
End Property

' Блоки информации
Private mobjFrame(7) As Object
Public Property Frame(ByVal FrameType As Tag_Info)
    Get
        Return mobjFrame(FrameType)
    End Get
    Set(ByVal Value)
        mobjFrame(FrameType) = Value
    End Set
End Property

' Извлекаем информацию из заданного файла
Public Sub Refresh()

    ' Таблица соответствия
    Dim strTag As New String(" ", 3)
    Dim strTitle As New String(" ", 30)
    Dim strArtist As New String(" ", 30)
    Dim strAlbum As New String(" ", 30)
    Dim strYear As New String(" ", 4)
    Dim strComment As New String(" ", 28)
    Dim bytDummy As Byte
    Dim bytTrack As Byte
    Dim bytGenre As Byte

    ' Открываем файл
    Dim intFile As Integer = FreeFile()
    FileOpen(intFile, strFileName, OpenMode.Binary, _
          OpenAccess.Read, OpenShare.LockWrite)

    ' Вычислим длину файла
    Dim lngLOF As Long = LOF(intFile)
    If (lngLOF > 128) Then

        ' Ищем тег ID3v1
        FileGet(intFile, strTag, lngLOF - 127, True)
        If (strTag.ToUpper <> "TAG") Then
```

```
            ' Тег ID3v1 не найден
            bTagExists = False
            mobjFrame(0) = ""
            mobjFrame(1) = ""
            mobjFrame(2) = ""
            mobjFrame(3) = ""
            mobjFrame(4) = ""
            mobjFrame(5) = ""
            mobjFrame(6) = ""

        Else

            ' ID3v1 нашли
            bTagExists = True

            ' Считываем блоки информации из файла
            FileGet(intFile, strTitle)
            FileGet(intFile, strArtist)
            FileGet(intFile, strAlbum)
            FileGet(intFile, strYear)
            FileGet(intFile, strComment)
            FileGet(intFile, bytDummy)
            FileGet(intFile, bytTrack)
            FileGet(intFile, bytGenre)

            ' Сопоставляем содержимое блоков свойствам
            mobjFrame(0) = strTitle
            mobjFrame(1) = strArtist
            mobjFrame(2) = strAlbum
            mobjFrame(3) = strYear
            mobjFrame(4) = bytTrack
            mobjFrame(5) = strComment
            mobjFrame(6) = bytGenre

        End If
    End If

    ' Закрываем файл
    FileClose(intFile)

End Sub
```

```
' Обновляем информацию
Public Sub Update()

    ' Таблица соответсвия
    Dim strTag As New String(" ", 3)
    Dim strTitle As New String(" ", 30)
    Dim strArtist As New String(" ", 30)
    Dim strAlbum As New String(" ", 30)
    Dim strYear As New String(" ", 4)
    Dim strComment As New String(" ", 28)
    Dim bytDummy As Byte
    Dim bytTrack As Byte
    Dim bytGenre As Byte

    ' Открываем файл
    Dim intFile As Integer = FreeFile()
    FileOpen(intFile, strFileName, OpenMode.Binary, _
       OpenAccess.ReadWrite, OpenShare.LockWrite)

    ' Получим длину файла
    Dim lngLOF As Long = LOF(intFile)
    If (lngLOF > 0) Then
        If (lngLOF > 128) Then

            ' Проверка на существования тега ID3v1
            FileGet(intFile, strTag, lngLOF - 127)
            If (strTag.ToUpper <> "TAG") Then

                ' Если нет тега, то добавим
                Seek(intFile, lngLOF)
                strTag = "TAG"
                FilePut(intFile, strTag)

            End If

            ' Выделим блоки для записи
            strTitle = LSet(mobjFrame(0), Len(strTitle))
            strArtist = LSet(mobjFrame(1), Len(strArtist))
            strAlbum = LSet(mobjFrame(2), Len(strAlbum))
```

```
                strYear = LSet(mobjFrame(3), Len(strYear))
                bytTrack = mobjFrame(4)
                strComment = LSet(mobjFrame(5), Len(strComment))
                bytGenre = mobjFrame(6)

                ' Записываем данные в файл
                FilePut(intFile, strTitle)
                FilePut(intFile, strArtist)
                FilePut(intFile, strAlbum)
                FilePut(intFile, strYear)
                FilePut(intFile, strComment)
                FilePut(intFile, bytDummy)
                FilePut(intFile, bytTrack)
                FilePut(intFile, bytGenre)

            End If
        End If

        ' Закрываем файл
        FileClose(intFile)

    End Sub
End Class
```

Коротко о том, что делает созданный класс. Во-первых, он обрабатывает имя файла. Если файл не существует, то выдается соответствующее предупреждение. Во-вторых, идет обработка файла. Считываются последние 128 байтов указанного файла и ищется специальная сигнатура. В случае обнаружения сигнатуры начинается дальнейшая обработка файла. Определяются различные данные песни путем сопоставления байтов таблице тегов. На этом обработка файла заканчивается.

Теперь у нас есть возможность либо извлекать информацию, заложенную в файлах MP3, либо изменять ее на новую. Это и реализовано в основной форме проекта. Добавьте на форму семь надписей `Label` с именами `lblTitle`, `lblArtist`, `lblAlbum`, `lblYear`, `lblTrack`, `lblComment`, `lblGenre`. Присвойте их свойствам `Text` соответствующие строки: `Название альбома`, `Артист`, `Альбом`, `Год`, `Трек`, `Комментарий`, `Жанр`. Теперь надо сопоставить установленным надписям шесть текстовых полей для первых шести надписей и одно поле со списком для надписи `Жанр`: `txtTitle`, `txtArtist`, `txtAlbum`, `txtYear`, `txtTrack`, `txtComment`, `cboGenre`. Удалите у всех добавленных эле-

ментов тексты по умолчанию. Теперь необходимо добавить две кнопки, с помощью которых мы будем считывать и записывать информацию: `butGetTag` и `butUpdate`. Свойству `Text` этих кнопок присвойте значения `Получить информацию` и `Изменить`. На этом визуальное построение проекта закончено. Приступаем к написанию кода. Откройте редактор кода. Сначала нам надо заполнить поле со списком названиями жанров. Сделаем это при загрузке формы (листинг 15.8).

Листинг 15.8. Получение информации из тегов файла

```
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        ' здесь привожу несколько жанров
        cboGenre.Items.Add("Blues")
        cboGenre.Items.Add("ClassicRock")
        cboGenre.Items.Add("Country")
        cboGenre.Items.Add("Dance")
        cboGenre.Items.Add("Disco")
        cboGenre.Items.Add("Funk")
        cboGenre.Items.Add("Grunge")
        cboGenre.Items.Add("Hip-Hop")
        cboGenre.Items.Add("Jazz")
        cboGenre.Items.Add("Metal")
        cboGenre.Items.Add("NewAge")
        cboGenre.Items.Add("Oldies")
        cboGenre.Items.Add("Other")
        cboGenre.Items.Add("Pop")
        ' продолжите список сами :-)
    End Sub


    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles butGetTag.Click
        Dim mp3 As New MP3TAG(Application.StartupPath & "..\..\cat.mp3")
        If (mp3.TagExists) Then
            txtTitle.Text = mp3.Frame(MP3TAG.Tag_Info.Title)
            txtArtist.Text = mp3.Frame(MP3TAG.Tag_Info.Artist)
            txtAlbum.Text = mp3.Frame(MP3TAG.Tag_Info.Album)
            txtYear.Text = mp3.Frame(MP3TAG.Tag_Info.Year)
            txtTrack.Text = MP3TAG.Tag_Info.Track
            txtComment.Text = mp3.Frame(MP3TAG.Tag_Info.Comment)
```

```
            cboGenre.SelectedIndex = mp3.Frame(MP3TAG.Tag_Info.Genre)
        End If
    End Sub

    Private Sub butUpdate_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles butUpdate.Click
        Dim mp3 As New MP3TAG("e:\down\18.mp3")
        mp3.Frame(MP3TAG.Tag_Info.Title) = txtTitle.Text
        mp3.Frame(MP3TAG.Tag_Info.Album) = txtAlbum.Text
        ' механизм записи номера жанра оставляю вам для самостоятельной
          работы
        mp3.Update()
    End Sub
```

Честно говоря, я устал вбивать все значения жанров в список. Думаю, что пора и вам немного постучать по клавишам. Список всех жанров я уже приводил выше. Для извлечения информации достаточно щелкнуть на кнопке **Получить информацию** (рис. 15.2).

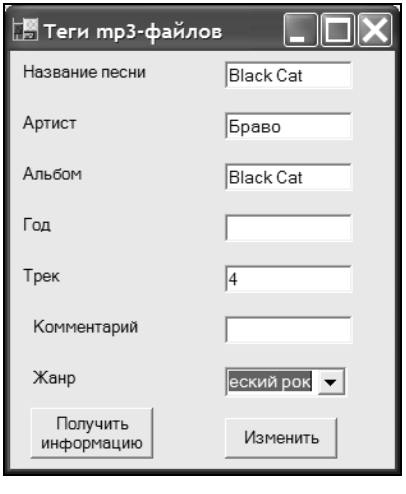

Рис. 15.2. Редактор тегов MP3

Сначала мы задаем имя музыкального файла. Для эксперимента я использовал песню "Черный кот" в исполнении группы "Браво". Эту песню в формате MP3 можно бесплатно загрузить с официального сайта группы по адресу **http://www.brawo.ru/**. Затем проверяем наличие сигнатуры в файле. Если в файле обнаруживается данная сигнатура, то начинаем извлекать нужную

информацию и аккуратно раскладывать по полочкам. Также возможен и обратный процесс. Если нам необходимо изменить информацию (или добавить), то открываем файл для записи и вписываем данные в нужном месте. Эта процедура показана в событии кнопки `butUpdate_Click`. В коде этого события отсутствует операция для записи номера жанра. Оставляю вам эту работу в качестве домашнего задания.

Ну вот, мы и научились извлекать или добавлять информацию в файл. Теперь вы можете создавать редактор тегов и встраивать его в свой музыкальный проигрыватель.

15.8. Советы и рекомендации

На сайте Microsoft был приведен пример создания виртуального пианино, написанный на VB 6.0 (файл Midismpl.exe). Данный пример весьма похож на проект, описанный в *разд. 15.5*. Только вместо файлов WAV программа генерировала MIDI-звуки. Попробуйте сами переделать тот пример для VB .NET.

Глава 16

Говорящие программы

В этой главе речь пойдет о создании говорящих приложений с помощью технологии синтеза и распознавания речи SAPI. На данный момент существуют две версии SAPI — четвертая и пятая, которые могут сосуществовать вместе. SAPI 4.0 широко используется в технологии Microsoft Agent 2.0, а SAPI 5.0 включена в состав операционных систем Windows XP и выше. Использование речевых движков в программах в последнее время стало очень модным. Многие приложения обзавелись возможностью проговаривать текст, создавая эффект живой программы.

16.1. Технология MS Agent 2.0

Речевые движки являются неотъемлемой частью технологии Microsoft Agent 2.0. Эта технология представляет собой целый комплекс различных продвинутых технологий, основанных на синтезе и распознавании речи. Но вершина ее — использование анимированных персонажей. Особенностью этих персонажей является их интерактивность, способность понимать команды с микрофона, воспроизводить заданный текст. Более подробно об этой технологии вы можете прочитать в моей книге, посвященной MS Agent[1]. Мы же с вами рассмотрим пример использования персонажей MS Agent в программах, написанных на VB .NET. Запустим новый проект `MSAgent`. Сначала надо подключить элемент `Microsoft Agent Control 2.0` к проекту. Делается это следующим образом. В меню **Project** выбираем команду **Add Reference**. Затем в открывшемся окне надо выбрать вкладку **COM** и найти в списке строку **Microsoft Agent Control 2.0**. Выделите найденную строку и нажмите кнопку **Select** (рис. 16.1).

[1] Климов А. П. MS Agent. Графические персонажи для интерфейсов. — СПб.: БХВ-Петербург, 2005.

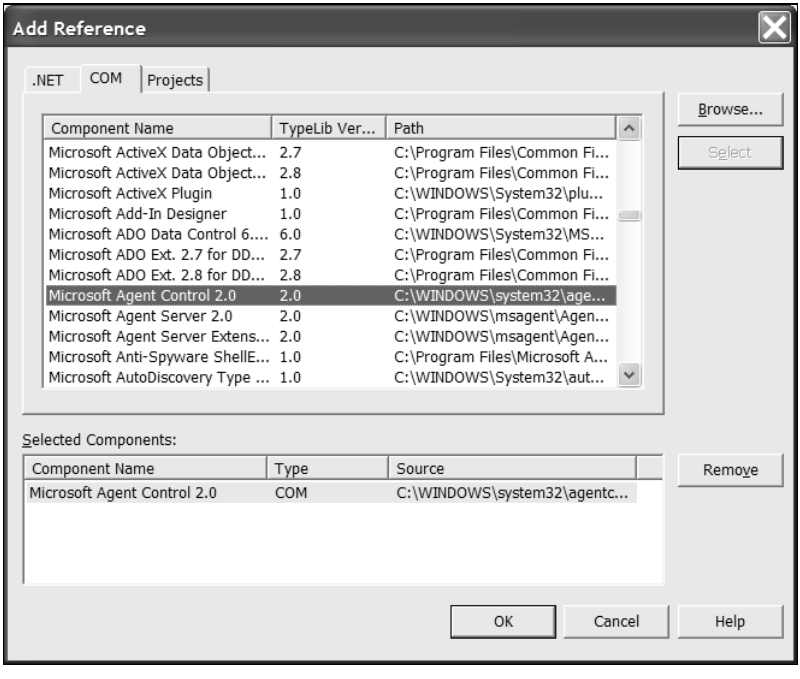

Рис. 16.1. Подключение MS Agent к проекту

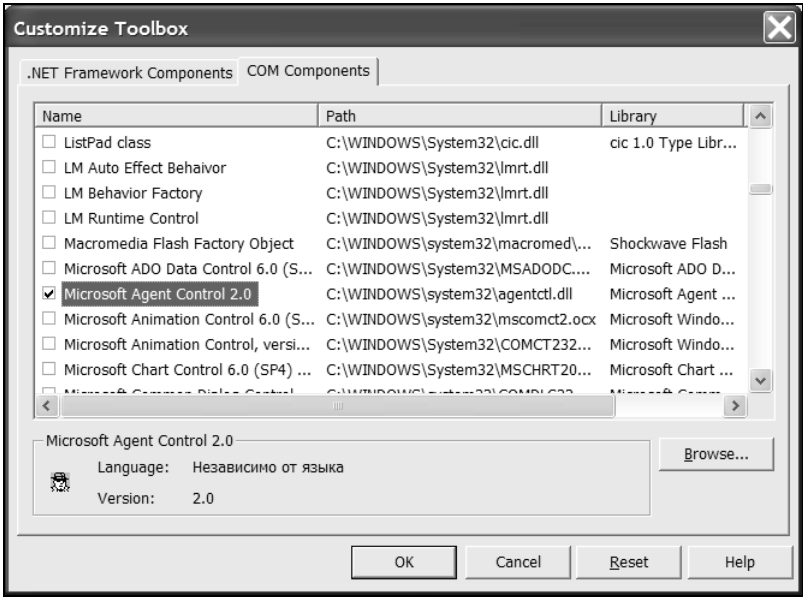

Рис. 16.2. Добавление элемента на панель инструментов

Далее необходимо добавить подключенный элемент на панель инструментов. В меню **Tools** выбираем команду **Add/Remove Tools Item** и в открывшемся диалоговом окне **Customize Toolbox** переходим на вкладку **COM Components**, устанавливаем флажок **Microsoft Agent Control 2.0** (рис. 16.2) и щелкаем на кнопке **OK**.

После описанных действий на вашей панели инструментов можно будет увидеть новый значок **Microsoft Agent Contor 2.0**. Поместите этот элемент на форму (рис. 16.3). По умолчанию ему будет присвоено имя `AxAgent1`.

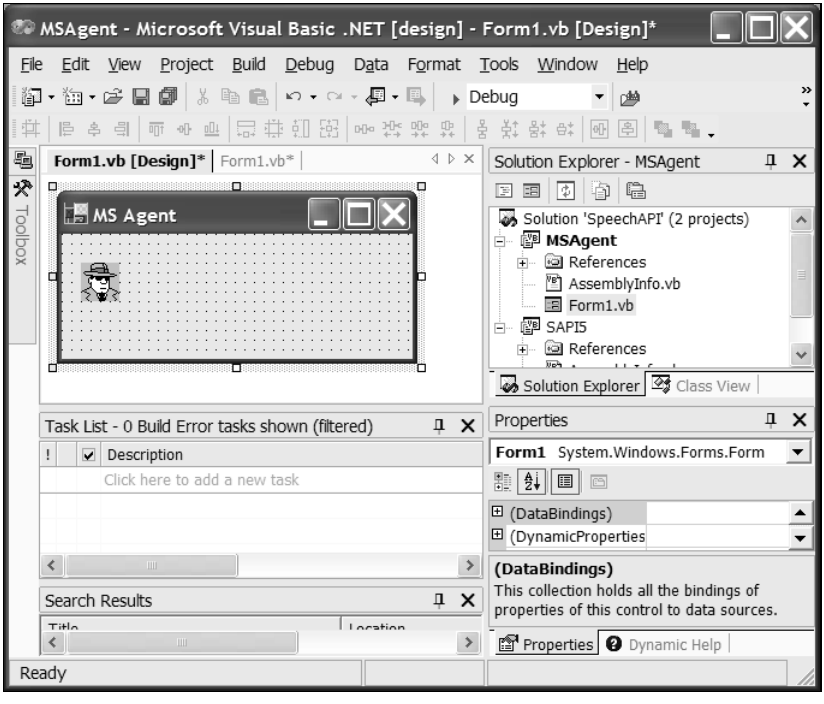

Рис. 16.3. Добавление элемента управления на форму

На этом этапе можно считать визуальное программирование законченным. Переходим в редактор кода и пишем следующий код (листинг 16.1).

Листинг 16.1. Использование MS Agent в программах

```
'--------------------------------------
' Использование MS Agent © 2004 Александр Климов
'--------------------------------------
Dim MyChar As AgentObjects.IAgentCtlCharacterEx
```

```
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        ' Загружаем персонаж
        AxAgent1.Characters.Load("CharacterID")
        MyChar = AxAgent1.Characters("CharacterID")
        ' Показываем персонаж
        MyChar.Show()
        ' Устанавливаем русский язык
        MyChar.LanguageID = &H419
        ' Устанавливаем женский голос
        MyChar.TTSModeID = "{06377F80-D48E-11d1-B17B-0020AFED142E}"
        ' Говорим
        MyChar.Speak("Здравствуйте, я ваша тетя!")
    End Sub
```

Запустите программу, и вы увидите анимированный персонаж, проговаривающий заданную фразу (рис. 16.4).

Рис. 16.4. Анимированный персонаж

Надо сразу предупредить, что внешний вид вашего персонажа может отличаться от помощника, представленного на рисунке. Дело в том, что я давно использую данную технологию в своей практике, и в моей коллекции более 100 различных персонажей на любой вкус. Вы тоже можете пополнить свою коллекцию агентов, посетив сайт **http://characters.narod.ru/.** По умолчанию у вас скорее всего установлен только один персонаж (для версий Windows ME/2000/XP и выше).

Нужно заметить, что персонажи могут не только выводить фразу в специальном окне, но и проговаривать ее голосом (через динамики). Но для этого необходимо установить несколько дополнительных файлов. В частности, чтобы персонаж заговорил по-русски, нужно установить соответствующий

голосовой движок Lernout & Hauspie TTS3000 TTS engine — Russian. А для владельцев машин под управлением Windows XP нужно также установить поддержку SAPI 4.0, несмотря на то, что в Windows XP уже имеется встроенная SAPI 5.1. Путаница с версиями SAPI вызвала волну недоумения на форумах, посвященных MS Agent. Регулярно каждый новый участник задает один и тот же вопрос: "Почему на моем компьютере под WinXP персонажи не говорят?" Эти две версии SAPI могут сосуществовать вместе в одной системе. Более подробную информацию об установке необходимых файлов вы можете посмотреть на странице **http://www.microsoft.com/msagent/downloads/user.asp**.

16.2. Два персонажа

Теперь, после короткого знакомства с MS Agent, можно перейти к более сложному примеру. В проекте `TwoAgents` мы рассмотрим возможности работы с несколькими персонажами, добавления команд в их контекстное меню, а также управления действиями помощников приказами, отдаваемыми через микрофон.

В своем примере я использовал двух котов (персонажи `Oscar` и `Pelotti`), которых скачал с сайта **http://characters.narod.ru** и скопировал в папку Windows/msagent/chars. Эта папка является стандартным хранилищем для персонажей. При хранении персонажей в стандартной папке отпадает необходимость прописывать полный путь к файлу персонажа, достаточно указать его имя. Если вы будете применять другие персонажи, то вам придется изменить имена персонажей в коде примера. В отличие от первого примера, мы уже не используем персонаж по умолчанию (что избавляло нас от необходимости указывать путь к файлу персонажа), а жестко указываем программе, с какими помощниками мы собираемся работать. Сначала приведу сам код (листинг 16.2).

Листинг 16.2. Использование двух персонажей MS Agent

```
'--------------------------------------
' Использование двух персонажей © 2004 Александр Климов
'--------------------------------------
    Dim Char1 As AgentObjects.IAgentCtlCharacterEx
    Dim Char2 As AgentObjects.IAgentCtlCharacterEx
    Dim Request As AgentObjects.IAgentCtlRequest

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        ' Загружаем персонажи
```

```
        AxAgent1.Characters.Load("pelotti", "pelotti.acs")
        Char1 = AxAgent1.Characters("pelotti")
        AxAgent1.Characters.Load("oscar", "oscar.acs")
        Char2 = AxAgent1.Characters("oscar")

        ' Выводим персонажи на экран в заданных точках
        Char1.Show()
        Char1.MoveTo(100, 200)
        Char2.Show()
        Char2.MoveTo(500, 200)

        ' Устанавливаем русский язык
        Char1.LanguageID = &H419
        Char2.LanguageID = &H419

        ' Добавляем команду Поздороваться. Звучит как "хеллоу"
        Char2.Commands.Add("Hi", "Поздороваться", "hello", True, True)

        ' Устанавливаем мужской и женский голоса
        Char1.TTSModeID = "{06377F81-D48E-11D1-B17B-0020AFED142E}"
        Char2.TTSModeID = "{06377F80-D48E-11d1-B17B-0020AFED142E}"

        ' Говорим
        Char2.Speak("Здравствуй, котяра! Чем занимался?")
        Request = Char2.Play("GestureRight")

        ' Ждем, когда Оскар покажет направо
        Char1.Wait(Request)

        ' Отвечаем
        Char1.Play("Congratulate")
        Request = Char1.Speak(txtSpeak.Text)

        Char2.Wait(Request)
        Request = Char2.Speak("Мой любимый мультфильм!\Mrk=1\")
    End Sub

    Private Sub butSpeak_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles butSpeak.Click
```

```
        ' Введите свой текст в текстовое поле
        Char1.Speak(txtSpeak.Text)

    End Sub

    Private Sub AxAgent1_Command(ByVal sender As Object, ByVal e As
AxAgentObjects._AgentEvents_CommandEvent) Handles AxAgent1.Command

        If e.userInput.name = "Hi" Then
            ' Если пользователь сказал Hello,
            ' то помахать лапой в ответ
            Char2.Play("Greet")
            Char2.Speak("Приветик, друг. Молочка не дашь?")
        End If
    End Sub

    Private Sub AxAgent1_Bookmark(ByVal sender As Object, ByVal e As
AxAgentObjects._AgentEvents_BookmarkEvent) Handles AxAgent1.Bookmark
        If e.bookmarkID = 1 Then
            Char2.LanguageID = &H409
        End If
    End Sub
```

Разберем написанный код. Не забудьте сначала повторить начальные действия из предыдущего примера — разместить элемент `AxAgent1` на форме. В самом начале мы объявляем несколько переменных. Переменные `Char1` и `Char2` являются объектами, содержащими свойства и методы персонажей. Объект `Request` позволяет добиться синхронности выполнения кода. Дело в том, что сами персонажи работают в асинхронном режиме. Представьте себе такую ситуацию. Вам необходимо воспроизвести некоторый диалог между двумя героями. Вы старательно, строчка за строчкой, выписали все реплики персонажей. Но, запустив программу, вы услышите совсем не то, на что надеялись. Ваши герои будут говорить одновременно, не дожидаясь ответов от оппонента.

```
' Выводим персонажей на экран в заданных точках
Char1.Show()
Char1.MoveTo(100, 200)
Char2.Show()
Char2.MoveTo(500, 200)
```

Итак, сначала мы выводим персонажи на экран. Метод `MoveTo` позволяет указать конкретное место расположения агента. Кстати, если вы хотите, чтобы персонажи сразу появлялись в заданной точке, то поместите вызов метода `MoveTo` перед вызовом метода `Show`. В нашем случае персонажи сначала появляются в верхнем левом углу экрана, а потом перемещаются в указанную точку.

```
' Устанавливаем русский язык
Char1.LanguageID = &H419
Char2.LanguageID = &H419
```

Не забываем установить соответствующий языковой идентификатор `LanguageID`, если предполагается вывод речи на русском языке.

```
' Добавляем команду Поздороваться. Звучит как "хеллоу"
Char2.Commands.Add("Hi", "Поздороваться", "hello", True, True)
```

Оператор `Add` позволяет добавлять собственные команды в контекстное меню персонажей, вызываемое правой кнопкой мыши. Я добавил команду `Поздороваться` — при выборе этой команды персонаж должен воспроизвести заданную анимацию. Надо сказать, что эта команда является альтернативой для голосовой команды `Hello`, если у вас нет микрофона. Если же микрофон у вас подключен, то вы можете отдавать команды вслух. Но об этом чуть позже.

Итак, мы определились с выбором языка. Далее у нас есть возможность выбора голоса — мужского или женского.

```
' Устанавливаем мужской и женский голоса
Char1.TTSModeID = "{06377F81-D48E-11D1-B17B-0020AFED142E}"
Char2.TTSModeID = "{06377F80-D48E-11d1-B17B-0020AFED142E}"
```

Конкретные значения для разных голосов вам придется искать в документации самостоятельно. Теперь выводим на экран первую фразу и воспроизводим анимацию `GestureRight` (персонаж указывает на левую часть экрана). Вот здесь нам и понадобится объект `Request`.

```
Request = Char2.Play("GestureRight")
```

Второй персонаж не будет предпринимать никаких действий, пока не наступит определенное нами событие. Как только указанное событие произошло, персонаж начнет выполнять свои движения. Обратите внимание, что мы снова используем объект `Request` для синхронизации действий (рис. 16.5).

```
Request = Char1.Speak(txtSpeak.Text)
Char2.Wait(Request)
Request = Char2.Speak("Мой любимый мультфильм!\Mrk=1\")
```

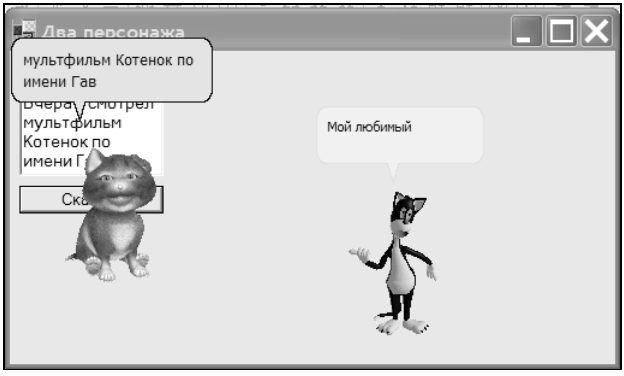

Рис. 16.5. Разговор двух персонажей

Существует еще одна возможность для синхронизации поведения агентов с помощью специальных закладок `Mrk`. Как только персонаж произнесет фразу, содержащую закладку, произойдет выполнение команды, описанной в этой закладке процедурой `AxAgent1.Bookmark`. Я поместил в закладку код, переключающий языковой идентификатор персонажа на английский язык. Сделано это для того, чтобы персонаж мог воспринимать ваши команды через микрофон. К великому сожалению, в настоящее время возможен только английский ввод речи. Реакция на голосовые команды задается в процедуре `AxAgent1.Command`.

Рис. 16.6. Персонаж пытается понять команду, отданную через микрофон

И, наконец, последнее в этой программе. Поместите на форме текстовое поле и кнопку. При нажатии на кнопку персонаж будет говорить текст, набранный в текстовом поле. Все готово для запуска программы. На ваших глазах будет разыгран небольшой спектакль с участием двух актеров.

Когда они наговорятся между собой, возьмите в руки микрофон, и, удерживая клавишу <Scroll Lock>, произнесите слово "Hello" (рис. 16.6). Если вы произнесли его правильно, персонаж распознает его и выполнит указанное действие.

В текстовом поле можно набирать любую фразу. После нажатия на кнопку вы услышите синтезированный голос.

Я рассказал вам только о малой толике возможностей технологии MS Agent. На основе двух описанных примеров вы можете создавать очень необычные программы, которые удивят пользователя своим нестандартным интерфейсом.

Анекдот в тему

Недалекое будущее. Компьютерная выставка. Идет презентация компьютера нового поколения. Менеджер компании, представляя новинку, говорит посетителям выставки:

– Этот компьютер уникален. Он не нуждается в вводе команд с клавиатуры, а воспринимает их с голоса пользователя. Сейчас каждый из вас может попробовать поуправлять этой чудо-машиной.

Тихий голос из толпы:

– Формат Цэ. Энтер!

16.3. Использование SAPI 5.1

Использование анимированных помощников в программе не всегда оправдано. Иногда требуется только возможность озвучить текст безо всякого визуального интерфейса. В Windows XP есть интересная программа Экранный диктор (**Пуск | Все программы | Стандартные | Специальные возможности**), которая голосом проговаривает различные сообщения.

Примечание

Кстати, любители MS Agent 2.0 придумали очень интересный трюк. Дело в том, что скрытый персонаж не способен говорить (так задумано разработчиками). Но кто мешает создать персонаж-невидимку? Был создан такой агент (точнее, пустое место), и программисты, привыкшие к MS Agent, просто используют в его программах. А пользователь даже не догадывается, что с ним говорит персонаж!

Итак, если нужен только озвученный текст, можно воспользоваться технологией синтеза речи от SAPI 5.1. Должен сказать, что у данного способа есть один весьма серьезный недостаток — отсутствие поддержки русского языка. Но, тем не менее, пример очень интересный, и пройти мимо него я не могу.

Для начала следует подключить к проекту необходимые компоненты (проект `SAPI5`). В меню **Project** выбираем команду **Add References**, переходим

в диалоговом окне на вкладку **COM** и ищем строку **Microsoft Voice Text**. Выделяем найденную строку и жмем на кнопку **Select**. После того как заданный объект выбран, нажимаем на кнопку **OK** и закрываем диалоговое окно. Щелкаем правой кнопкой на панели инструментов и выбираем команду **Add/Remove Items**. В открывшемся диалоговом окне снова переходим на вкладку **COM** и устанавливаем флажок **TextToSpech Class**. Нажимаем **OK**, чтобы закрыть диалоговое окно. Теперь на панели инструментов в разделе **Components** появится новый компонент с таким же названием с изображением губ.

Поместите полученный компонент на форму. Результат получится немного неожиданным — на окне появится большой полураскрытый рот! Вы можете сразу присвоить свойству `Visible` значение `False`, если не желаете видеть этот рот во время выполнения программы. В нашем примере мы не будем его скрывать, так как нас ожидает интересное зрелище — во время вывода текста губы будут синхронно шевелиться, создавая иллюзию говорящего человека. Весь код выполняется в обработчике события для кнопки (листинг 16.3).

Рис. 16.7. Вид программы с говорящим ртом

Листинг 16.3. Говорящий рот

```
'--------------------------------------
' Говорящий рот © 2005 Александр Климов
'--------------------------------------
```

```
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
        AxTextToSpeech1.Speak("I love cats!")
    End Sub
```

Запустите программу, включите динамики на компьютере и нажмите на кнопку. Вы должны услышать мое признание в любви (рис. 16.7).

16.4. Советы и рекомендации

Синтез речи находит широкое применение при создании аудиокниг, справочной документации, образовательных игр для детей. Для более полной информации по этой теме обратитесь к документации на странице **http://msdn.microsoft.com**.

Глава 17

Законы природы

Окружающие нас предметы подчиняются законам природы — падает ли на голову яблоко, улетает ли воздушный шарик в небо или скачет мячик по земле. Однажды, когда я еще учился в школе, моя учительница по физике дала нам домашнее задание — написать сочинение на тему: "Один день без силы трения". Фантазия учеников разыгралась не на шутку — от сползающих частей одежды до невозможности затормозить машину. Урок, на котором читались наши сочинения, был самым веселым уроком по физике за все годы. Но на самом деле, реальные последствия после исчезновения силы трения трудно представить.

Как ни парадоксально, написать программу для компьютера, моделирующую тот или иной закон природы, гораздо проще. Вы должны понимать, что изучение подобных законов — не праздное любопытство, а необходимое условие для написания динамических игр. Такие игры, как пинг-понг, летный симулятор, арканоид, основаны на применении сил упругости, гравитации, свободного падения и других законов. В этой главе мы разберем несколько типичных примеров использования законов природы.

17.1. Отражение

Отражение фигур от стенки — весьма популярный прием в создании игр. Наиболее часто он применяется в аркадных играх. Самыми яркими примерами использования отражения фигур от стенки являются такие игры, как пинг-понг и бильярд. Итак, наша задача — научиться писать программы, в которых фигура (обычно это шар) отражается от стенки и движется в другую сторону, подчиняясь определенным физическим законам.

Как правило, в книгах сразу описывается готовый алгоритм отражения от стенок. Но подобный подход имеет крупный недостаток. Программист не получает необходимых навыков программирования. И когда ему понадобится решить ту или иную сходную задачу, он теряется, не владея общей методикой решения сложных задач. Я предлагаю вам другой подход. При поста-

новке сложного задания постарайтесь упростить его до самого возможного минимума. Затем вводите новые условия, и постепенно вы придете к искомой цели. Давайте разложим задачку на отражение от стенок по полочкам, начиная с самых азов.

Создаем новый проект `Bounce`. Поместим на форму надпись `Label` и присвоим элементу имя `lblBall`. Это наше первое упрощение. Обычно в программах рисуется окружность, выполняющая роль мячика. Но здесь кроются подводные камни, которые нам только помешают сосредоточиться на понимании техники отражения. Использование готового объекта облегчит нам жизнь. Для большого сходства с мячиком присвойте свойству `Text` значение `O`, а также увеличьте размер шрифта и цвет фона:

- `Name` = `lblBall`;
- `AutoSize` = `True`;
- `BackColor` = `Red`;
- `Font.Size` = `20`;
- `Text` = `O`.

Теперь необходимо добавить на форму таймер и кнопку, запускающую движение нашего импровизированного мячика. Установите им следующие свойства:

- `Name` = `butStart`;
- `Text` = `Старт`;
- `Name` = `tmrBounce`;
- `Interval` = `50`.

Визуальная часть программы готова. Переходим к кодированию. Самое простое, что можно сделать с мячиком, это запустить его в какую-нибудь сторону. Давайте отправим его в левую сторону. Обратите внимание, что мы бросаем мяч строго по горизонтали (помните наше правило — максимально упростить задачу). Тогда можно предложить следующее решение (листинг 17.1).

Листинг 17.1. Смещение объекта в сторону

```
'--------------------------------------
' Отражения © 2004 Александр Климов
'--------------------------------------
   Private Sub butStart_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles butStart.Click
        tmrBounce.Enabled = True
    End Sub
```

```
    Private Sub tmrBounce_Tick(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles tmrBounce.Tick
        lblBall.Left = lblBall.Left - 1
    End Sub
```

Код весьма примитивный. Щелчок на кнопке активизирует таймер, который, в свою очередь, сдвигает объект на 1 пиксел влево. Теперь для самопроверки самостоятельно попробуйте переписать код, чтобы импровизированный мячик двигался в правую сторону, вверх и вниз. Надеюсь, этот этап вы прошли благополучно. Чуть-чуть усложняем задачу. В нашей программе мяч улетает далеко влево, исчезая за пределами нашей формы. Давайте принудительно остановим его при достижении левого края формы (листинг 17.2).

Листинг 17.2. Остановка движущегося объекта

```
    Private Sub tmrBounce_Tick(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles tmrBounce.Tick
        If lblBall.Left <= 0 Then
            tmrBounce.Enabled = False
        End If
        lblBall.Left = lblBall.Left - 1

    End Sub
```

Здесь мы ввели проверку на достижение мячиком левого края формы, и при наступлении данного события останавливаем таймер. А теперь — внимание! Нам нужно научиться отражать объект от стенки. Как можно видеть, при обратном движении идет не убавление координат объекта, а их приращение. На языке математики это означает, что вектор движения поменял свой знак на противоположный. Нам понадобится новая переменная на уровне класса `gDirection`, которая и будет отвечать за направление движения мячика. Этот флаг будет принимать два значения — положительное или отрицательное. Используя значение флага, мы всегда будем знать, в каком направлении нужно двигать объект. Но сначала инициализируем флаг:

```
Dim gDirection As Integer = 1
```

А в коде для события `tmrBounce_Tick` изменим одну строчку:

```
lblBall.Left = lblBall.Left - 1 * gDirection
```

Как только объект достигает края формы (`If lblBall.Left <= 0`), мы меняем значение флага `gDirection` на противоположное, тем самым заставляя увеличивать значения координат объекта. Запустите проект (листинг 17.3), чтобы убедиться в этом. Как вы уже догадались, для отражения мячика от

правого края формы используется схожий прием (не забудьте о ширине самого мячика).

Листинг 17.3. Отражение объекта от стенок

```
    Private Sub tmrBounce_Tick(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles tmrBounce.Tick
        lblBall.Left = lblBall.Left - 1 * gDirection

        ' Отскок от левого края формы
        If lblBall.Left <= 0 Then
            gDirection = -1
        End If

        ' Отскок от правого края формы. Учитываем ширину мячика
        If lblBall.Left >= ClientSize.Width - lblBall.Width Then
            gDirection = 1
        End If
    End Sub
```

17.2. Отражение под углом

Мы научились отражать объекты от стенок при условии, что они движутся к ним строго перпендикулярно. Но в реальной жизни объекты двигаются под некоторыми углами. Таким образом, нам надо научиться отражать фигуры под углом. Запустим новый проект `Bounce2` и перепишем из предыдущего примера весь код, который послужит нам шаблоном. Итак, когда мы отражали мячик строго по горизонтали (или вертикали), то у нас менялись координаты мяча вдоль одной оси координат. Отражения под углом задействуют уже две оси координат. Таким образом, нам нужно менять сразу два значения свойств у объекта — `Top` и `Left`.

Давайте рассмотрим этот процесс поподробнее. Возьмем простейший случай. Пусть мячик отражается от стенок формы. Если внимательно посмотреть на траекторию движения мяча, то можно заметить, что при отражении от боковых стенок меняется на противоположное значение координата `X` (`Left`) — если она увеличивалась, то после отскока начинает уменьшаться. Аналогично при отражении от горизонтальных сторон прямоугольника меняется на противоположное значение координата `Y` (`Top`). Суммируя эти наблюдения, делаем вывод, что одной глобальной переменной для направления движения мяча уже недостаточно. Удаляем переменную `gDirection` и

создаем две новые глобальные переменные gDirX и gDirY. Соответственно, для таймера переписываем код, обрабатывая сразу все четыре возможных варианта отскока мяча от стенок формы (листинг 17.4).

Листинг 17.4. Отражения объекта под углом

```
'--------------------------------------
' Отражения под углом © 2004 Александр Климов
'--------------------------------------
    Dim gDirX As Integer = 1
    Dim gDirY As Integer = 1

    Private Sub tmrBounce_Tick(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles tmrBounce.Tick
        lblBall.Left = lblBall.Left - 5 * gDirX
        lblBall.Top = lblBall.Top - 5 * gDirY

        ' Отскок от левого края формы
        If lblBall.Left <= 0 Then
            gDirX = -1
        End If

        ' Отскок от правого края формы. Учитываем ширину мячика
        If lblBall.Left >= ClientSize.Width - lblBall.Width Then
            gDirX = 1
        End If

        ' Отскок от верхнего края формы
        If lblBall.Top <= 0 Then
            gDirY = -1
        End If

        ' Отскок от нижнего края формы. Учитываем ширину мячика
        If lblBall.Top >= ClientSize.Height - lblBall.Height Then
            gDirY = 1
        End If
    End Sub
```

```
    Private Sub butStart_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles butStart.Click
        tmrBounce.Enabled = True
    End Sub
```

17.3. Отражение настоящего мячика

На практике никто не использует в качестве объектов для отражения кнопки, надписи, списки или другие элементы управления. Они предназначены совсем для других целей. На самом деле при написании игр программисты прибегают к помощи фигур, предоставляемых классом `Graphics`, — проект `RealBall` (листинг 17.5).

Листинг 17.5. Отражение с использованием графических объектов

```
'--------------------------------------
' Отражение настоящего мячика © 2004 Александр Климов
'--------------------------------------
    ' Текущие координаты мяча
    Private xCenter, yCenter As Integer

    ' Размеры мяча
    Private xBallSize As Integer = 40
    Private yBallSize As Integer = 40

    ' Направление движения мяча
    Dim gDirX, gDirY As Integer

    Dim cxTotal, cyTotal As Integer
    Private bm As Bitmap

    Private Sub Form1_Resize(ByVal sender As Object, ByVal e As
System.EventArgs) Handles MyBase.Resize
        Dim g As Graphics = CreateGraphics()

        gDirX = 3
        gDirY = 3
        cxTotal = (xBallSize + 2 * gDirX)
        cyTotal = (yBallSize + 2 * gDirY)
```

```
        bm = New Bitmap(cxTotal, cyTotal)
        g = Graphics.FromImage(bm)
        g.Clear(BackColor)
        DrawBall(g, New Rectangle(gDirX, gDirY, _
                                  xBallSize, yBallSize))
        g.Dispose()

        ' Помещаем мяч в центр формы
        xCenter = ClientSize.Width \ 2
        yCenter = ClientSize.Height \ 2
    End Sub

    Protected Overridable Sub DrawBall(ByVal g As Graphics, _
                                       ByVal rect As Rectangle)
        ' Рисуем мяч
        g.FillEllipse(Brushes.Plum, rect)
    End Sub

    Private Sub Timer1_Tick(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Timer1.Tick
        Dim g As Graphics = CreateGraphics()
        g.DrawImage(bm, xCenter - cxTotal \ 2, _
                        yCenter - cyTotal \ 2, cxTotal, cyTotal)
        g.Dispose()

        xCenter += gDirX
        yCenter += gDirY

        ' Меняем направления движения при достижении любой стенки
        If (xCenter + xBallSize / 2 >= ClientSize.Width) OrElse _
           (xCenter - xBallSize / 2 <= 0) Then
            gDirX = -gDirX
        End If

        If (yCenter + yBallSize / 2 >= ClientSize.Height) OrElse _
           (yCenter - yBallSize / 2 <= 0) Then
            gDirY = -gDirY
        End If
    End Sub
```

Теперь наш пример больше похож на настоящую программу, которую можно использовать как основу для создания игр, связанных с отражением объектов от стенок.

17.4. Отскоки

Отскок является частным случаем отражения. По сути, это отражение предмета с затуханием. Если поднять мяч над головой и отпустить его, то, подчиняясь закону тяготения, мяч упадет на землю, подскочит обратно вверх (сила упругости) и снова начнет свое движение вниз. Причем каждый раз мяч будет подниматься на меньшую высоту, чем при предыдущем отскоке. Для упрощения будем считать, что мяч при этом всегда будет падать в одну и ту же точку. Снова для наших опытов воспользуемся обычной кнопкой.

Примечание

Безусловно, прямоугольная кнопка по форме весьма отдаленно напоминает мяч. Точнее, совсем не напоминает. Но для нас важно понять сам принцип, описывающий отскок предметов от поверхности.

В реальности отскакивающий объект не поднимается на прежнюю высоту, теряя часть своей энергии. Значит, нам нужна дополнительная переменная, которая будет управлять затуханием отскока. Для примера нам понадобятся две кнопки, а также один таймер `Timer1`. Весь код, управляющий отскоками кнопки, сосредоточен в событии таймера `Tick` — проект `FallBall` (листинг 17.6).

Листинг 17.6. Отскок мяча от поверхности

```
'--------------------------------------
' Отскоки © 2004 Александр Климов
'--------------------------------------
    ' Текущая позиция объекта
    Dim cy As Integer
    ' Нижняя граница формы
    Dim floorY As Integer = ClientSize.Height
    ' Скорость падения
    Dim ymov As Integer
    ' Сила притяжения
    Dim gravity As Integer = 1
    ' Коэффициент затухания
    Dim decay As Single = 0.9
```

```
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
        ' Возвращаем кнопку в верхнюю часть формы
        Button2.Top = 10
        ' Включаем таймер
        Timer1.Enabled = True
    End Sub

    Private Sub Timer1_Tick(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Timer1.Tick

        ymov += gravity
        cy = Button2.Top + ymov ' Падаем вниз
        Button2.Top = cy

        If Button2.Height + Button2.Top >= floorY Then
            ' Если достигли нижней границы формы,
            ' то отскакиваем вверх, меняя направление движения
            ' с затуханием скорости
            ymov *= -1 * decay
            Button2.Top = floorY - Button2.Height
        End If
    End Sub
```

Перейдем к разбору примера. Вначале я объявил несколько переменных на уровне класса, которые отвечают за направление движения, скорость и силу затухания отскока объекта. А дальше все просто. При нажатии первой кнопки мы запускаем таймер — объект начинает падать под силой собственной тяжести (имитация гравитации):

```
ymov += gravity
cy = Button2.Top + ymov ' Падаем вниз
Button2.Top = cy
```

По достижении объектом нижней границы формы мы меняем направление движения и уменьшаем силу отскока с помощью переменной `decay`. Вы можете управлять силой отскока мяча, изменяя значение этой переменной. При значениях, близких к 0, мяч будет падать камнем вниз без малейшего отскока. Приведенный пример я постарался упростить до минимума. Возможно, для решения ваших собственных задач вам придется внести в пример некоторые изменения.

17.5. Советы и рекомендации

В этой короткой главе я рассмотрел только самые простые примеры, основанные на различных законах природы. Вспомните, какие еще существуют законы, и смоделируйте их на компьютере. Например, изобразите траекторию движения ядра, выпущенного из пушки. Как известно, снаряд летит по параболе. Вы можете предусмотреть возможность изменения угла выстрела пушки, тем самым изменяя дальность полета снаряда и место его падения. Как видите, написание подобных примеров требует от вас знания математики и физики. И если вы в дальнейшем собираетесь сосредоточиться на написании игр, то вам просто необходимо полюбить точные науки.

Глава 18

Игры

Кто сам программирует свои компьютерные игры,
тот наслаждается дважды.

Жак Арсак

Написать свою игру, наверное, пробовал любой программист. Программирование игры является одной из самых популярных тем, интересных как начинающему, так и опытному программисту. Создание компьютерных игр предоставляет очень полезную возможность повышения своей квалификации. Программирование игр подразумевает очень хорошее знание не только самого языка программирования, но также и математики, физики, графики и т. п. В этой главе будет показано, как написать несколько своих игр. Хочу сразу предупредить, что тема программирования игр настолько обширна, что можно написать целую книгу. Объем данной книги просто не позволит мне рассмотреть все аспекты, связанные с играми. Поэтому в этой главе будут рассмотрены самые простые реализации известных игр, знакомых вам с детства. Моя цель в данном случае — пробудить интерес к программированию игр и показать, что это совсем не сложно.

18.1. Сейф. Вариант первый

В свое время была очень популярна игра "Братья Пилоты. По следам полосатого слона", выполненная в жанре квест. И вот как-то раз я застрял на одном уровне. Там надо было вскрыть сейф. Система замка выглядела следующим образом. На двери имелось 16 ручек, расположенных в виде 4 на 4. Часть ручек была в вертикальном положении, другая — в горизонтальном. При повороте любой одной ручки все ручки из одного с ней ряда и столбца меняли свое положение на противоположное. На этом уровне я застрял надолго. Здесь же задержались и многие другие пользователи, которые засыпали игровые форумы вопросами о вскрытии сейфа. Каким-то чудом мне удалось подобрать правильную комбинацию и пройти дальше. Но осадок неудовлетворенности остался. И мне захотелось воссоздать такую ситуацию собственными силами, написав этот эпизод самостоятельно. Но в то время я

только осваивал программирование и боялся приступать к реализации своей задачи. И вдруг на бескрайних просторах Интернета мне попалась страничка с этой игрой, написанная на VBScript! А скриптовый язык VBScript является упрощенным подобием языка Visual Basic. И мне не составило большого труда переписать эту игру под Visual Basic 5.0. Простота кода позволила без проблем перенести его и на VB 6.0, и на Visual Basic .NET 2003. К сожалению, у меня не сохранилось имя автора, написавшего тот скрипт к игре. Но, тем не менее, огромное ему спасибо! Прежде чем мы будем писать свою первую игру, давайте познакомимся с ее HTML-версией (листинг 18.1).

Листинг 18.1. Игра "Сейф" на VBScript

```
<HTML><HEAD><TITLE>БРАТЬЯ ПИЛОТЫ</TITLE>
<META http-equiv=Content-Type content="text/html;
charset=windows-1251">
<STYLE type=text/css>
.button {
       BORDER-TOP-WIDTH: 5px; FONT-WEIGHT: bold;
BORDER-LEFT-WIDTH: 5px; FONT-SIZE: 30px; BORDER-LEFT-COLOR:
steelblue; BACKGROUND: darkslategray; BORDER-BOTTOM-WIDTH: 5px;
BORDER-BOTTOM-COLOR: steelblue; WIDTH: 50px; COLOR: #ffffff;
BORDER-TOP-COLOR: steelblue; HEIGHT: 50px; BORDER-RIGHT-WIDTH:
5px; BORDER-RIGHT-COLOR: steelblue
}
.button1 {
       BORDER-TOP-WIDTH: 7px; FONT-WEIGHT: bold;
BORDER-LEFT-WIDTH: 7px; FONT-SIZE: 13px; BORDER-LEFT-COLOR:
steelblue; BACKGROUND: steelblue; BORDER-BOTTOM-WIDTH: 7px;
BORDER-BOTTOM-COLOR: steelblue; WIDTH: 100px; COLOR: #ffffff;
BORDER-TOP-COLOR: steelblue; HEIGHT: 35px; BORDER-RIGHT-WIDTH:
7px; BORDER-RIGHT-COLOR: steelblue
}
</STYLE>
</HEAD>
<BODY>
<H2>БРАТЬЯ ПИЛОТЫ</H2>
  <FORM name=game>
<div id="start">
  <P>Игра по мотивам квеста БРАТЬЯ ПИЛОТЫ. Ваша
```

```
цель - щелкая на ячейках, сделать так, чтобы во всех их появились
точки. Жмите кнопку Старт и играйте.</P></DIV>


<DL>
  <DD align="center">
  <TABLE cellSpacing=0 cellPadding=0 width=200 border=0>
    <TBODY>
    <TR>
      <TD width=100><TT><INPUT class=button1 type=button value=СТАРТ
       name=start></TT>
      </TD>
      <TD width=100><TT><INPUT class=button1 type=button
       name=ochki></TT>
    </TD></TR></TBODY></TABLE>
  <TABLE id=pole height=200 cellSpacing=0 cellPadding=0 width=200
   border=0>
    <TBODY>
    <TR>
      <TD width=50><INPUT class=button type=button name=but1></TD>
      <TD width=50><INPUT class=button type=button name=but2></TD>
      <TD width=50><INPUT class=button type=button name=but3></TD>
      <TD width=50><INPUT class=button type=button name=but4></TD></TR>
    <TR>
      <TD width=50><INPUT class=button type=button name=but5></TD>
      <TD width=50><INPUT class=button type=button name=but6></TD>
      <TD width=50><INPUT class=button type=button name=but7></TD>
      <TD width=50><INPUT class=button type=button name=but8></TD></TR>
    <TR>
      <TD width=50><INPUT class=button type=button name=but9></TD>
      <TD width=50><INPUT class=button type=button name=but10></TD>
      <TD width=50><INPUT class=button type=button name=but11></TD>
      <TD width=50><INPUT class=button type=button
       name=but12></TD></TR>
    <TR>
      <TD width=50><INPUT class=button type=button name=but13></TD>
      <TD width=50><INPUT class=button type=button name=but14></TD>
      <TD width=50><INPUT class=button type=button name=but15></TD>
      <TD
  width=50><INPUT class=button type=button
  name=but16></TD></TR></TBODY></TABLE></FORM>
```

```
    <SCRIPT language=VBScript>
<!--
' игра Инверсия, по мотивам одной из задач в игре БРАТЬЯ ПИЛОТЫ
dim c(16)
dim a(16)
dim s
dim nn
 Randomize
pole.style.visibility = "hidden"
start.style.visibility = ""
sub start_onclick
s=0
       for i=1 to 16
       c(i)=int(rnd(1)+0.5)
        if c(i)=0 then a(i)="0" else a(i)="."
       next
pole.style.visibility = ""
start.style.visibility = "hidden"
 call hod
end sub
sub ochki_onclick
       nn = nn + 1
       if nn >= 10 then
              alert "И не надоело щелкать? Лучше займитесь делом!"
              nn = 0
       end if
end sub
sub hod
document.game.but1.value=a(1)
document.game.but2.value=a(2)
document.game.but3.value=a(3)
document.game.but4.value=a(4)
document.game.but5.value=a(5)
document.game.but6.value=a(6)
document.game.but7.value=a(7)
document.game.but8.value=a(8)
document.game.but9.value=a(9)
document.game.but10.value=a(10)
document.game.but11.value=a(11)
```

```
document.game.but12.value=a(12)
document.game.but13.value=a(13)
document.game.but14.value=a(14)
document.game.but15.value=a(15)
document.game.but16.value=a(16)
f=0
for i=1 to 16
if a(i)="." then f=f+1
next
if f=16 then call winer
s=s+1
document.game.ochki.value=s-1 & " ходов"
end sub
sub winer
document.game.but1.value="П"
document.game.but2.value="О"
document.game.but6.value="Б"
document.game.but7.value="Е"
document.game.but11.value="Д"
document.game.but12.value="А"
document.game.but13.value="!"
document.game.but14.value="!"
document.game.but15.value="!"
document.game.but16.value="!"
end sub

sub but1_onclick
 for i=1 to 4
 if a(i)="0" then a(i)="." else a(i)="0"
 next
  for i=5 to 13 step 4
  if a(i)="0" then a(i)="." else a(i)="0"
  next
 call hod
end sub

sub but2_onclick
 for i=1 to 4
 if a(i)="0" then a(i)="." else a(i)="0"
```

```
 next
  for i=6 to 14 step 4
  if a(i)="0" then a(i)="." else a(i)="0"
  next
 call hod
end sub

sub but3_onclick
 for i=1 to 4
 if a(i)="0" then a(i)="." else a(i)="0"
 next
  for i=7 to 15 step 4
  if a(i)="0" then a(i)="." else a(i)="0"
  next
 call hod
end sub

sub but4_onclick
 for i=1 to 4
 if a(i)="0" then a(i)="." else a(i)="0"
 next
  for i=8 to 16 step 4
  if a(i)="0" then a(i)="." else a(i)="0"
  next
 call hod
end sub

sub but5_onclick
 for i=5 to 8
 if a(i)="0" then a(i)="." else a(i)="0"
 next
  if a(1)="0" then a(1)="." else a(1)="0"
  if a(9)="0" then a(9)="." else a(9)="0"
  if a(13)="0" then a(13)="." else a(13)="0"
 call hod
end sub

sub but6_onclick
 for i=5 to 8
```

```
 if a(i)="0" then a(i)="." else a(i)="0"
 next
  if a(2)="0" then a(2)="." else a(2)="0"
  if a(10)="0" then a(10)="." else a(10)="0"
  if a(14)="0" then a(14)="." else a(14)="0"
 call hod
end sub

sub but7_onclick
 for i=5 to 8
 if a(i)="0" then a(i)="." else a(i)="0"
 next
  if a(3)="0" then a(3)="." else a(3)="0"
  if a(11)="0" then a(11)="." else a(11)="0"
  if a(15)="0" then a(15)="." else a(15)="0"
 call hod
end sub

sub but8_onclick
 for i=5 to 8
 if a(i)="0" then a(i)="." else a(i)="0"
 next
  if a(4)="0" then a(4)="." else a(4)="0"
  if a(12)="0" then a(12)="." else a(12)="0"
  if a(16)="0" then a(16)="." else a(16)="0"
 call hod
end sub

sub but9_onclick
 for i=9 to 12
 if a(i)="0" then a(i)="." else a(i)="0"
 next
  if a(1)="0" then a(1)="." else a(1)="0"
  if a(5)="0" then a(5)="." else a(5)="0"
  if a(13)="0" then a(13)="." else a(13)="0"
 call hod
end sub

sub but10_onclick
 for i=9 to 12
```

```
 if a(i)="0" then a(i)="." else a(i)="0"
 next
  if a(2)="0" then a(2)="." else a(2)="0"
  if a(6)="0" then a(6)="." else a(6)="0"
  if a(14)="0" then a(14)="." else a(14)="0"
 call hod
end sub

sub but11_onclick
 for i=9 to 12
 if a(i)="0" then a(i)="." else a(i)="0"
 next
  if a(3)="0" then a(3)="." else a(3)="0"
  if a(7)="0" then a(7)="." else a(7)="0"
  if a(15)="0" then a(15)="." else a(15)="0"
 call hod
end sub

sub but12_onclick
 for i=9 to 12
 if a(i)="0" then a(i)="." else a(i)="0"
 next
  if a(4)="0" then a(4)="." else a(4)="0"
  if a(8)="0" then a(8)="." else a(8)="0"
  if a(16)="0" then a(16)="." else a(16)="0"
 call hod
end sub

sub but13_onclick
 for i=13 to 16
 if a(i)="0" then a(i)="." else a(i)="0"
 next
  for i=1 to 9 step 4
  if a(i)="0" then a(i)="." else a(i)="0"
  next
 call hod
end sub

sub but14_onclick
 for i=13 to 16
```

```
 if a(i)="0" then a(i)="." else a(i)="0"
 next
  for i=2 to 10 step 4
  if a(i)="0" then a(i)="." else a(i)="0"
  next
 call hod
end sub

sub but15_onclick
 for i=13 to 16
 if a(i)="0" then a(i)="." else a(i)="0"
 next
  for i=3 to 11 step 4
  if a(i)="0" then a(i)="." else a(i)="0"
  next
 call hod
end sub

sub but16_onclick
 for i=13 to 16
 if a(i)="0" then a(i)="." else a(i)="0"
 next
  for i=4 to 12 step 4
  if a(i)="0" then a(i)="." else a(i)="0"
  next
 call hod
end sub
-->
</SCRIPT>
</dl>
</BODY></HTML>
```

При желании вы можете воспроизвести эту игру. Создайте текстовый файл, скопируйте текст из листинга и переименуйте файл в safe.htm (этот файл представлен и на компакт-диске). После этого вы сможете играть в нее, запустив свой браузер Internet Explorer (рис. 18.1).

Но наша задача — перенести данную игру с VBScript на язык Visual Basic .NET 2003. Я намеренно не стал для первого примера сильно оптимизировать программу. Цель упражнения — показать, что запрограммировать игру

под силу даже начинающему программисту, еще слабо разбирающемуся во многих конструкциях языка. Запускаем VB. NET для нового проекта `Safe`. Сначала мы просто размещаем на форме 16 кнопок. Будьте аккуратны, располагайте кнопки друг за другом по порядку по четыре кнопки в ряд. Рядом с ними поместите еще одну кнопку с надписью **Старт** и надпись `Label`, свойству `Text` которой присвойте значение `0`. Теперь просто переписываем код из VBScript. Изменения практически минимальны (листинг 18.2).

Рис. 18.1. Игра, написанная на VBScript

Листинг 18.2. Пример по мотивам игры "Братья Пилоты"

```
'---------------------------------------
' Открывание сейфа © 2005 Александр Климов
'---------------------------------------
```

```
Dim c(16) As Integer
Dim a(16) As String
Dim s As Integer
Dim i As Integer
Dim f As Integer

Private Sub Hod()
    Button1.Text = a(1)
    Button2.Text = a(2)
    Button3.Text = a(3)
    Button4.Text = a(4)
    Button5.Text = a(5)
    Button6.Text = a(6)
    Button7.Text = a(7)
    Button8.Text = a(8)
    Button9.Text = a(9)
    Button10.Text = a(10)
    Button11.Text = a(11)
    Button12.Text = a(12)
    Button13.Text = a(13)
    Button14.Text = a(14)
    Button15.Text = a(15)
    Button16.Text = a(16)
    s = s + 1
    Label1.Text = Str(s - 1)
    f = 0
    For i = 1 To 16
        If a(i) = "." Then
            f = f + 1
        End If
    Next
    If f = 16 Then
        Call Winer()
    End If
End Sub

' Процедура Winner() просто выводит надпись Победа на кнопках
Private Sub Winer()
    Button1.Text = "П"
```

```
        Button2.Text = "О"
        Button6.Text = "Б"
        Button7.Text = "Е"
        Button11.Text = "Д"
        Button12.Text = "А"
        Button13.Text = "!"
        Button14.Text = "!"
        Button15.Text = "!"
        Button16.Text = "!"
End Sub


Private Sub butStart_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles butStart.Click
        s = 0
        For i = 1 To 16
            c(i) = Int(Rnd(1) + 0.5)
            Randomize()
            If c(i) = 0 Then
                a(i) = "0"
            Else : a(i) = "."
            End If
        Next


        Call Hod()
End Sub


Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
        For i = 1 To 4
            If a(i) = "0" Then
                a(i) = "."
            Else
                a(i) = "0"
            End If
        Next
        For i = 5 To 13 Step 4
            If a(i) = "0" Then
                a(i) = "."
            Else
```

```
                a(i) = "0"
            End If
        Next
        Call Hod()
End Sub
```

Я не стал приводить код для всех кнопок, так как он абсолютно идентичен коду из примера на VBScript, поэтому листинг содержит код только для первой кнопки. Вначале мы запускаем цикл и присваиваем массиву `c` случайные значения от `0` до `1`. Если значение элемента массива будет равно `0`, то кнопке присваивается значение `0`. В противном случае кнопке присваивается значение "." Теперь только осталось для каждой кнопки написать код, который просто меняет значения кнопок из ее ряда и столбца на противоположный. Все, игра готова! Можете запускать и играть (рис. 18.2). Чем хорош данный пример? Прежде всего своей простотой. Используя такой подход, вы легко сможете переписать эту же игру, скажем, на Java или C#. Но все-таки надо признать, что предложенный пример далек от идеала. Код слишком громоздкий и неуклюжий. Давайте перепишем игру заново с учетом возможностей, предоставляемых Visual Basic .NET 2003.

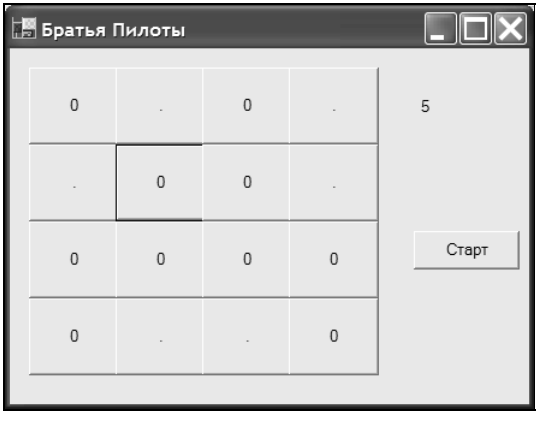

Рис.18.2. Игра "Сейф". Первый вариант

18.2. Второй вариант игры "Сейф"

Теперь мы попробуем написать игру более грамотно с технической точки зрения. Во-первых, не будем вручную создавать все 16 кнопок и расставлять их на форме. Это слишком утомительно и неудобно. Вспомните игру "Сапер",

входящую в состав Windows. Там игровое поле может состоять из сотен кнопок. Вы представляете, сколько может занять времени простая расстановка кнопок? Мы поручим эту работу компьютеру и создадим массив кнопок программным путем. Кроме того, внесем небольшие косметические изменения в программу, чтобы она выглядела более эстетично для пользователя. Запускаем VB .NET и создаем новый проект `Safe2`. На этот раз мы ничего не размещаем на форме, а сразу переходим в редактор кода (листинг 18.3).

Листинг 18.3. Второй вариант игры

```
'--------------------------------------
' Открывание сейфа-2 © 2005 Александр Климов
'--------------------------------------
Private Const BUTTONS16 As Integer = 16
Private SQUARE_ROOT As Integer = _
   Convert.ToInt32(Math.Sqrt(BUTTONS16))
Private btn(BUTTONS16 - 1) As Button


    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        ' Задаем свойства для формы
        MyBase.MaximizeBox = False
        MyBase.FormBorderStyle = FormBorderStyle.Fixed3D
        MyBase.Text = "Сейф-2"

        Dim i As Integer
        Dim w As Integer = MyBase.ClientSize.Width / SQUARE_ROOT
        Dim h As Integer = MyBase.ClientSize.Height / SQUARE_ROOT
        For i = 0 To BUTTONS16 - 1
            btn(i) = New Button
            btn(i).Width = w
            btn(i).Height = MyBase.ClientSize.Height / SQUARE_ROOT
            btn(i).Left = (i Mod SQUARE_ROOT) * w
            btn(i).Top = (i \ SQUARE_ROOT) * h
            btn(i).BackColor = Color.SkyBlue
            AddHandler btn(i).Click, AddressOf btn_Click
            btn(i).Tag = 0
            MyBase.Controls.Add(btn(i))
        Next

    End Sub
```

```
Private Sub ChangeButtonState(ByVal i As Integer)
    If (Convert.ToInt32(btn(i).Tag) = 0) Then
        btn(i).Tag = 1
        btn(i).BackColor = Color.Red
    Else
        btn(i).Tag = 0
        btn(i).BackColor = Color.SkyBlue
    End If
End Sub
Public Sub btn_Click(ByVal sender As Object, ByVal e As EventArgs)

    If Equals(sender, btn(0)) Then
        ChangeButtonState(0)
        ChangeButtonState(1)
        ChangeButtonState(2)
        ChangeButtonState(3)
        ChangeButtonState(4)
        ChangeButtonState(8)
        ChangeButtonState(12)
    End If
    ' для второй кнопки
    If Equals(sender, btn(1)) Then
        ChangeButtonState(0)
        ChangeButtonState(1)
        ChangeButtonState(2)
        ChangeButtonState(3)
        ChangeButtonState(5)
        ChangeButtonState(9)
        ChangeButtonState(13)
    End If
    'для третьей кнопки
    If Equals(sender, btn(2)) Then
        ChangeButtonState(0)
        ChangeButtonState(1)
        ChangeButtonState(2)
        ChangeButtonState(3)
        ChangeButtonState(6)
        ChangeButtonState(10)
```

```
    ChangeButtonState(14)
End If
'для четвертой кнопки
If Equals(sender, btn(3)) Then
    ChangeButtonState(0)
    ChangeButtonState(1)
    ChangeButtonState(2)
    ChangeButtonState(3)
    ChangeButtonState(7)
    ChangeButtonState(11)
    ChangeButtonState(15)
End If
'для пятой кнопки
If Equals(sender, btn(4)) Then
    ChangeButtonState(0)
    ChangeButtonState(4)
    ChangeButtonState(5)
    ChangeButtonState(6)
    ChangeButtonState(7)
    ChangeButtonState(8)
    ChangeButtonState(12)
End If
'для шестой кнопки
If Equals(sender, btn(5)) Then
    ChangeButtonState(1)
    ChangeButtonState(4)
    ChangeButtonState(5)
    ChangeButtonState(6)
    ChangeButtonState(7)
    ChangeButtonState(9)
    ChangeButtonState(13)
End If
'для седьмой кнопки
If Equals(sender, btn(6)) Then
    ChangeButtonState(2)
    ChangeButtonState(4)
    ChangeButtonState(5)
    ChangeButtonState(6)
    ChangeButtonState(7)
```

```
        ChangeButtonState(10)
        ChangeButtonState(14)
    End If
    If Equals(sender, btn(7)) Then
        ChangeButtonState(3)
        ChangeButtonState(4)
        ChangeButtonState(5)
        ChangeButtonState(6)
        ChangeButtonState(7)
        ChangeButtonState(11)
        ChangeButtonState(15)
    End If
    If Equals(sender, btn(8)) Then
        ChangeButtonState(0)
        ChangeButtonState(4)
        ChangeButtonState(8)
        ChangeButtonState(9)
        ChangeButtonState(10)
        ChangeButtonState(11)
        ChangeButtonState(12)
    End If
    If Equals(sender, btn(9)) Then
        ChangeButtonState(1)
        ChangeButtonState(5)
        ChangeButtonState(8)
        ChangeButtonState(9)
        ChangeButtonState(10)
        ChangeButtonState(11)
        ChangeButtonState(13)
    End If
    If Equals(sender, btn(10)) Then
        ChangeButtonState(2)
        ChangeButtonState(6)
        ChangeButtonState(8)
        ChangeButtonState(9)
        ChangeButtonState(10)
        ChangeButtonState(11)
        ChangeButtonState(14)
    End If
```

```
        If Equals(sender, btn(11)) Then
            ChangeButtonState(3)
            ChangeButtonState(7)
            ChangeButtonState(8)
            ChangeButtonState(9)
            ChangeButtonState(10)
            ChangeButtonState(11)
            ChangeButtonState(15)
        End If
        If Equals(sender, btn(12)) Then
            ChangeButtonState(0)
            ChangeButtonState(4)
            ChangeButtonState(8)
            ChangeButtonState(12)
            ChangeButtonState(13)
            ChangeButtonState(14)
            ChangeButtonState(15)
        End If
        If Equals(sender, btn(13)) Then
            ChangeButtonState(1)
            ChangeButtonState(5)
            ChangeButtonState(9)
            ChangeButtonState(12)
            ChangeButtonState(13)
            ChangeButtonState(14)
            ChangeButtonState(15)
        End If
        If Equals(sender, btn(14)) Then
            ChangeButtonState(2)
            ChangeButtonState(6)
            ChangeButtonState(10)
            ChangeButtonState(12)
            ChangeButtonState(13)
            ChangeButtonState(14)
            ChangeButtonState(15)
        End If
        If Equals(sender, btn(15)) Then
            ChangeButtonState(3)
            ChangeButtonState(7)
```

```
            ChangeButtonState(11)
            ChangeButtonState(12)
            ChangeButtonState(13)
            ChangeButtonState(14)
            ChangeButtonState(15)
        End If

        If CheckForWin() Then Congratulation()
    End Sub
    Private Function CheckForWin() As Boolean
        Dim i As Integer
        For i = 0 To BUTTONS16 - 1
            If (Convert.ToInt32(btn(i).Tag) = 0) Then Return False
        Next
        Return True
    End Function
    Private Sub Congratulation()
        MessageBox.Show("Поздравляю!!!")
        Dim i As Integer
        For i = 0 To BUTTONS16 - 1
            btn(i).BackColor = Color.SkyBlue
            btn(i).Tag = 0
        Next
    End Sub

    Private Sub mnuStart_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles mnuStart.Click
        Dim i As Integer
        Dim c(15)
        For i = 0 To 15
            c(i) = Int(Rnd(1) + 0.5)
            Randomize()
            If c(i) = 0 Then
                btn(i).BackColor = Color.SkyBlue
                btn(i).Tag = 0
            Else
                btn(i).BackColor = Color.Red
```

```
                btn(i).Tag = 1
            End If
        Next
    End Sub
```

Мы объявили константу `BUTTONS16`, содержащую количество используемых в игре кнопок. Переменную `SQUARE_ROOT` можно рассматривать как пространство, занимаемое одной кнопкой. В массиве `btn` будут содержаться объекты типа `Button`, что позволит нам легко обращаться к различным свойствам любой кнопки и писать для нее обработчик событий. В момент загрузки формы (событие `Load`) я задал свойства для формы и кнопок. Как видите, мы вычисляем в цикле размеры одной кнопки и с помощью несложного алгоритма располагаем их в квадрате 4×4. Одновременно все кнопки получают одинаковый цвет, и в свойство `Tag` записывается `0`. Вместо неинтересных надписей на кнопках мы будем менять цвета самих кнопок. Такое решение более подходит для игр. Эта концепция реализована через процедуру `ChangeButtonState`. Теперь осталось только написать обработчик для события `Click` всех 16 кнопок. Мы уже добавили в программе обработчик событий `btn_Click` с помощью оператора `AddHandler`. Напишем код, который будет менять цвета кнопок согласно правилам игры. Я пошел по пути наименьшего сопротивления и прописал код отдельно для каждой кнопки. Возможно, вам удастся найти более компактное решение и оптимизировать пример.

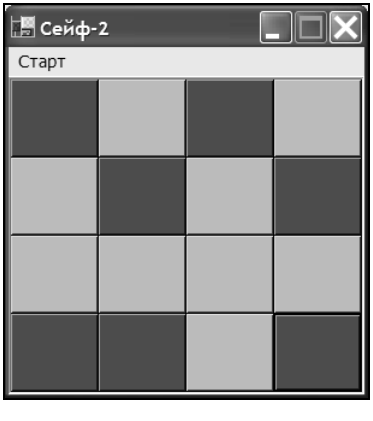

Рис. 18.3. Игра "Сейф". Второй вариант

Последние две процедуры `CheckForWin()` и `Congratulation()` служат для определения выигрышной комбинации и выводят сообщение с поздравле-

нием. Игра практически готова. Единственный недостаток — отсутствие инициализации игры. При запуске все квадраты окрашены одним цветом. Следует добавить какой-то код, случайным образом устанавливающий цвета. Это можно сделать в событии `Form_Load`, а также с помощью дополнительной кнопки **Старт** или командой меню (рис. 18.3). Как вы заметили, я не стал в этом примере считать количество сделанных ходов. Думаю, вы и сами справитесь с таким заданием. Кроме того, в конце главы вы найдете еще несколько советов по улучшению игры. Удачи вам!

18.3. Крестики-нолики

Вероятно, эту игру знают все. И, тем не менее, стоит всегда сформулировать задачи, которые предстоит решать перед непосредственным программированием игры. Я рассматриваю "пляжный" вариант игры. Под этим я понимаю максимальное приближение процесса игры к реальной жизни. То есть поле расчерчивается палкой на песке двумя вертикальными и горизонтальными линиями. И игроки, передавая по очереди палку друг другу, чертят свои крестики и нолики. Иными словами, речь идет о варианте Игрок-Игрок. Вариант Игрок-Компьютер в данном примере не рассматривается.

18.3.1. Теория игры

Итак, в игре два игрока по очереди делают ходы в расчерченном поле 3 на 3 квадрата. Один из игроков ставит в свободное поле крестик, другой — нолик. Перед игрой игроки должны договориться, кто ходит первым, и каким знаком он будет отмечать свой ход. Цель игры — первым выстроить прямую линию из своих знаков по вертикали, горизонтали или по диагонали. Если пустых клеток на поле больше нет, и никто из игроков не смог выстроить линию, то игра завершилась вничью.

18.3.2. Визуальное представление игры

Запускаем Visual Basic .NET 2003 и создаем новый проект `XO`. Присвоим свойству `Text` формы значение `Крестики-нолики`. Так как мы реализуем пляжный вариант, то нам понадобятся четыре линии. Но в VB .NET 2003 уже нет элементов управления `Line`. Теперь все линии на форме рисуются программно. Однако если вы ветеран программирования на Visual Basic, и привычка рисовать линии визуальным способом слишком сильна, то можно выйти из положения с помощью элементов управления `Label`. Для этого добавляем на форму четыре элемента `Label`, чтобы нарисовать сетку 3×3. Удалите у них текст, а свойству `BackColor` присвойте какой-нибудь цвет. Я присвоил им цвет `ActiveCaption` из вкладки **System**. Обратите внимание,

что вы не сможете сделать линии в редакторе формы слишком тонкими с помощью мыши. Установите минимально возможную толщину линий в редакторе формы и перейдите в редактор свойств. Там найдите пункт **Size** и щелкните на плюсике. Теперь вручную установите нужные параметры. Для горизонтальных линий это будет высота (**Height**), а для вертикальных — ширина (**Width**). Я присвоил им значения 4. Далее, для игры нам понадобятся несколько элементов для отображения символов **X** и **O**, которыми мы будем отмечать ходы игроков. В нашей созданной сетке находятся 9 пустых клеток. Расположим в этих местах кнопки. Выбор кнопок обусловлен тем, что у нас появится возможность сосредоточиться на программировании игры, поручив выполнять многие операции самой среде программирования. В начале игры все поля должны быть пустыми, поэтому удалим у всех кнопок из свойства `Text` содержащийся текст. В принципе, нам неважно, кто будет ходить первым, а кто вторым. Мы не будем перегружать игру лишними элементами для подсчета выигрышей и прочей чепухой, а сосредоточимся на игре. Единственное, что, пожалуй, можно добавить, — это кнопка **Старт** для запуска новой игры.

18.3.3. Написание кода

Теперь можно приступать к кодированию. Понятно, что все кнопки должны совершать одинаковые операции — при нажатии на кнопку должен появиться символ **X** или **O**, а сама кнопка стать недействительной. Это предотвратит возможность повторного нажатия на кнопку. Поэтому нет смысла писать отдельно код для каждой кнопки. Позже я покажу, как распространить одинаковый код на все кнопки. А пока давайте сначала напишем код для первой кнопки (листинг 18.4).

Листинг 18.4. Код для первой кнопки

```
'--------------------------------------
' Крестики-нолики © 2005 Александр Климов
'--------------------------------------
Private Sub Button1_Click(ByVal sender As Object, ByVal e As
System.EventArgs) Handles Button1.Click


        If flag Then
            sender.Text = "0"
            flag = False
        Else
            sender.Text = "X"
```

```
            flag = True
        End If
        sender.Enabled = False
End Sub
```

Переменная `flag` — это глобальная переменная, принимающая два значения `True` или `False`. Ее надо объявить после строчки `Inherits System.Windows.Forms.Form`:

```
Dim flag As Boolean
```

В зависимости от флага на кнопке появляется символ **X** или **O**. По сути, флаг просто последовательно меняет очередь игроков. Сначала играет игрок **X**, затем игрок **O**.

Запустите проект и нажмите первую кнопку. Если нет ошибок, то вы увидите, что на кнопке появился символ **O**, а сама кнопка стала недоступной. Закрываем программу и возвращаемся в редактор кода. Так как все кнопки выполняют те же операции, что и первая кнопка, то нет необходимости писать повторно этот же код для каждой кнопки. А теперь — внимание! В VB .NET можно очень просто присоединить остальные кнопки к обработчику события первой кнопки через запятую в параметре `Handles`:

```
Private Sub Button1_Click(ByVal sender As Object, ByVal e As
System.EventArgs) Handles Button1.Click, Button2.Click, Button3.Click,
Button4.Click, Button5.Click, Button6.Click, Button7.Click,
Button8.Click, Button9.Click
```

Снова запускаем проект для проверки. Убеждаемся, что у нас нет ошибок и в целом все работает. Единственный недостаток — программа не определяет победителя. Сами игроки должны решать, кто выиграл очередную партию, и начать новую игру. К счастью, я благоразумно добавил кнопку **Старт**, запускающую игру снова. Ее назначение — очистить текст у всех кнопок и сделать эти кнопки доступными для нажимания (листинг 18.5).

Листинг 18.5. Код для старта игры

```
    Private Sub btnStart_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnStart.Click
        Button1.Enabled = True
        Button1.Text = ""
        Button2.Enabled = True
        Button2.Text = ""
        Button3.Enabled = True
        Button3.Text = ""
```

```
        Button4.Enabled = True
        Button4.Text = ""
        Button5.Enabled = True
        Button5.Text = ""
        Button6.Enabled = True
        Button6.Text = ""
        Button7.Enabled = True
        Button7.Text = ""
        Button8.Enabled = True
        Button8.Text = ""
        Button9.Enabled = True
        Button9.Text = ""
    End Sub
```

Снова запускаем программу и проверяем работу кнопки **Старт**. Как видите, теперь мы можем в любой момент начать новую игру. Но программа по-прежнему не может определять победителя. Ну, почему у нас такой глупый компьютер? Возможно, вам приходилось помогать в приобретении компьютера своим знакомым, которые в них весьма слабо разбираются. Такие знакомые уверены, что любой только что купленный компьютер — это такая умная машина, которая уже знает ответы на все вопросы и не требует установки каких-то операционных систем и различного рода программ. Подобную уверенность они почерпнули после просмотра голливудских блокбастеров, где главные герои задают главному компьютеру каверзные вопросы, а тот, не задумываясь, дает правильный ответ. Впрочем, мы отвлеклись. Если не полениться и посчитать, сколько всего существует выигрышных ситуаций, то вы, к удивлению, обнаружите, что их всего восемь. Три по горизонтали, три по вертикали и два по диагонали. Не так уж много для умной машины. Пишем процедуру, которая будет обнаруживать выигрышную комбинацию путем проверки текста кнопок. Процедура должна проверять все возможные комбинации и, в случае выигрыша одним из игроков, выводить соответствующее сообщение (листинг 18.6).

Листинг 18.6. Код на выявление выигрышной ситуации

```
    Private Sub CheckWin()
        If (Button1.Text = Button2.Text And Button1.Text = Button3.Text
        And Button1.Text <> "") _
        Or (Button4.Text = Button5.Text And Button4.Text = Button6.Text
        And Button4.Text <> "") _
```

```
        Or (Button7.Text = Button8.Text And Button7.Text = Button9.Text
        And Button7.Text <> "") _
        Or (Button1.Text = Button4.Text And Button1.Text = Button7.Text
        And Button1.Text <> "") _
        Or (Button2.Text = Button5.Text And Button2.Text = Button8.Text
        And Button2.Text <> "") _
        Or (Button3.Text = Button6.Text And Button3.Text = Button9.Text
        And Button3.Text <> "") _
        Or (Button1.Text = Button5.Text And Button1.Text = Button9.Text
        And Button1.Text <> "") _
        Or (Button3.Text = Button5.Text And Button3.Text = Button7.Text
        And Button3.Text <> "") Then
            Dim Str As String
            If flag Then
                Str = "X"
            Else
                Str = "O"
            End If
            MsgBox(UCase(Str) & " выиграл", MsgBoxStyle.OKOnly)

        End If
        If counter = 9 Then
            MsgBox("Ничья")
        End If
    End Sub
```

Добавляем вызов написанной процедуры в конце обработчика события кнопки `Button1.Click`:

```
...
Call CheckWin()
```

На 99 процентов игра сделана. Запускайте и играйте на здоровье (рис. 18.4).

Если вам повезет, то вы даже не поймете, почему программа на самом деле не закончена. Это главная проблема всех создателей игр. Порой они забывают о какой-нибудь неучтенной детали и выпускают игру в свет. А противные пользователи потом забрасывают разработчика письмами (в лучшем случае) с криками о помощи. На моей памяти был такой случай. Одно время, когда покемономания была в самом разгаре, я скачал одну игру с участием Пикачу (одного из самых любимых покемонов у детей). Так вот, в этой игре ему нужно было уворачиваться от бросаемого в него мяча. Я не-

сколько дней пытался играть по правилам и постоянно проигрывал, так как мяч летел по немыслимой траектории и попадал в моего покемона. Потом я записал эту игру одной девочке, и она прошла все уровни за полчаса. Оказывается, достаточно было просто подвести Пикачу вплотную к врагу, и мячи благополучно пролетали мимо него. И игра теряла всякий смысл.

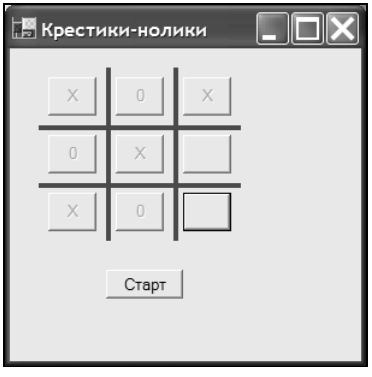

Рис. 18.4. Игра "Крестики-нолики"

Вернемся к нашей игре. Кроме выигрышной ситуации в игре возможна и ничья! Решить эту проблему очень легко. Если хорошенько подумать, то ничья получится только в одном случае — должны быть заполнены все клетки-кнопки игрового поля (в нашем случае их девять). Таким образом, достаточно добавить новую переменную-счетчик и прибавлять после каждого хода к счетчику единичку. Когда счетчик достигнет значения 9, значит, мы попали в ничейную ситуацию, и нужно показать соответствующее сообщение игрокам. Переменную-счетчик `counter` объявим как глобальную после строчки, где объявлена переменная `flag`:

```
Dim counter As Integer = 0
```

Для подсчета количества ходов добавляем код в обработчик кнопки `Button1.Click`:

```
counter += 1
```

Введенный код просто прибавляет единицу к предыдущему значению. И наконец, в процедуре `CheckWin()` добавляем еще одно условие:

```
If counter = 9 Then
            MsgBox("Ничья")
 End If
```

Только, пожалуйста, не забудьте сбросить показания счетчика при новой игре, иначе разгневанные письма от игроков полетят и к вам:

```
Private Sub btnStart_Click...
...
 counter = 0
End Sub
```

Вот теперь, кажется, все.

18.3.4. Крестики-нолики с интеллектом

Но не всегда есть возможность играть по сети или на одном компьютере с человеком. Тогда роль соперника поручается самому компьютеру. Однако чтобы машина играла с вами на равных, надо наделить ее каким-то интеллектом. Тема разработки искусственного интеллекта в играх очень сложна и требует отдельного разговора. Мы же попытаемся еще раз написать игру "Крестики-нолики" с небольшими зачатками ума у компьютера (проект `TicTacToe`). Для разнообразия я буду использовать другую технику создания игры. Написание программы можно разделить на три части: модель игры, графический интерфейс и поведение противника (искусственный интеллект).

Интерфейс и модель игры

Начнем с самого простого. Создадим внешний вид программы. На этот раз игровое поле будет состоять не из кнопок, а из девяти надписей `Label`. Расположим их в квадрате 3×3 и удалим из них текст. Так как цвет надписей совпадает с фоном формы, то изменим цвет надписей на любой другой для контраста. Имена надписям дадим хитрым образом. Они должны будут отражать в своих именах свои координаты на игровом поле. Координаты всех девяти надписей могут быть представлены как следующая таблица:

0,0 0,1 0,2

1,0 1,1 1,2

2,0 2,1 2,2

В соответствии с этой таблицей и дадим имена: `lbl00`, `lbl01`, `lbl02`, `lbl10`, `lbl11`. Оставим пока в покое нашу форму и перейдем к написанию кода, описывающему модель игры. Для этого будем использовать новый класс `GameModel.vb`. Щелкните правой кнопкой на имени проекта в окне проектов и выберите команду **Add | Add Class**. В диалоговом окне **Add New Item** введите новое имя класса `GameModel.vb` вместо стандартного `Class.vb` и нажмите на кнопку **Open**. Откроется редактор кода для созданного класса.

В данном классе у нас будет содержаться модель игры: свойства, перечисления, события (листинг 18.7).

Листинг 18.7. Модель игры "Крестики-нолики 2"

```
'--------------------------------------
' Крестики-нолики 2 © 2004 Александр Климов
'--------------------------------------
Public Class GameModel
    Private m_Square(2, 2) As SquareStatus
    Private m_Winner As SquareStatus

    ' Признак победы
    Private bVictrory As Boolean

    ' Событие для выигрышной комбинации
    Public Event Win(ByVal Winner As SquareStatus)
    ' Событие для ничьи
    Public Event Draw()

    ' Три возможных состояния в клетке
    Public Enum SquareStatus
        Empty = 0
        PlayerX = 1
        PlayerO = 2
    End Enum

    Public Sub New()
        Me.Clear()
    End Sub

    ' Очистим поле игры
    Public Sub Clear()
        Dim iRow As Integer, iCol As Integer
        For iRow = 0 To 2
            For iCol = 0 To 2
                m_Square(iRow, iCol) = SquareStatus.Empty
            Next
        Next
```

```
        bVictrory = False
        m_Winner = SquareStatus.Empty
    End Sub

    Public Property Square(ByVal iRow As Integer, ByVal iCol As Integer)
As SquareStatus
        Get
            Return m_Square(iRow, iCol)
        End Get
        Set(ByVal Value As SquareStatus)
            If m_Square(iRow, iCol) = SquareStatus.Empty And _
               bVictrory = False Then
                m_Square(iRow, iCol) = Value
                m_Winner = CheckWin()
                If m_Winner = SquareStatus.Empty Then
                    If CheckDraw() = True Then
                        RaiseEvent Draw()
                    End If
                Else
                    bVictrory = True
                    RaiseEvent Win(m_Winner)
                End If
            End If
        End Set
    End Property

    Public Property Square(ByVal SC As SquareCoordinate) As SquareStatus
        Get
            Return m_Square(SC.Row, SC.Column)
        End Get
        Set(ByVal Value As SquareStatus)
            Me.Square(SC.Row, SC.Column) = Value
        End Set
    End Property


    Private Function CheckWin() As SquareStatus
        Dim iTemp As SquareStatus
```

```
        iTemp = CheckForHorizontalWin()
        If iTemp <> SquareStatus.Empty Then Return iTemp

        iTemp = CheckForVerticalWin()
        If iTemp <> SquareStatus.Empty Then Return iTemp

        Return CheckForDiagonalWin()
    End Function

    Private Function CheckForHorizontalWin() As SquareStatus
        Dim iRow As Integer

        Dim iFirstSquarePlayer As SquareStatus

        For iRow = 0 To 2
            iFirstSquarePlayer = m_Square(iRow, 0)
            If iFirstSquarePlayer = m_Square(iRow, 1) And _
               iFirstSquarePlayer = m_Square(iRow, 2) Then
                Return iFirstSquarePlayer
            End If
        Next

        Return SquareStatus.Empty
    End Function

    Private Function CheckForVerticalWin() As SquareStatus
        Dim iCol As Integer
        Dim iFirstSquarePlayer As SquareStatus

        For iCol = 0 To 2
            iFirstSquarePlayer = m_Square(0, iCol)
            If iFirstSquarePlayer = m_Square(1, iCol) And _
               iFirstSquarePlayer = m_Square(2, iCol) Then
                Return iFirstSquarePlayer
            End If
        Next

        Return SquareStatus.Empty
    End Function
```

```
    Private Function CheckForDiagonalWin() As SquareStatus
        Dim iCol As Integer
        Dim iRow As Integer
        Dim iFirstSquarePlayer As SquareStatus

        iFirstSquarePlayer = m_Square(1, 1)

        If iFirstSquarePlayer = m_Square(0, 0) And _
           iFirstSquarePlayer = m_Square(2, 2) Then
            Return iFirstSquarePlayer
        End If

        If iFirstSquarePlayer = m_Square(0, 2) And _
           iFirstSquarePlayer = m_Square(2, 0) Then
            Return iFirstSquarePlayer
        End If
    End Function

    Private Function CheckDraw() As Boolean
        Dim iRow As Integer, iCol As Integer
        For iRow = 0 To 2
            For iCol = 0 To 2
                If m_Square(iRow, iCol) = SquareStatus.Empty Then Return
                False
            Next
        Next

        Return True
    End Function

End Class
Public Class SquareCoordinate
    Private m_Row As Integer
    Private m_Col As Integer

    Public Sub New()
        m_Row = 0 : m_Col = 0
    End Sub
```

```
    Public Sub New(ByVal iRow As Integer, ByVal iCol As Integer)
        m_Row = iRow : m_Col = iCol
    End Sub

    Public Property Row()
        Get
            Return m_Row
        End Get
        Set(ByVal Value)
            m_Row = Value
        End Set
    End Property

    Public Property Column()
        Get
            Return m_Col
        End Get
        Set(ByVal Value)
            m_Col = Value
        End Set
    End Property

    Public Overrides Function ToString() As String
        Return " [" & m_Row & ", " & m_Col & "] "
    End Function
End Class
```

Итак, каждое поле игры может иметь три состояния: пустая клетка, **X** и **O**. Поместим эти состояния в перечисление `SquareStatus`. Игровое поле, таким образом, состоит из квадратов 3×3, где каждому квадрату `m_Square` соответствует одно из значений `SquareStatus`. Следующий шаг — создание свойства `Square`, которое будет получать или устанавливать состояние игрового поля `SquareStatus`. Игрок может устанавливать либо крестик **X**, либо нолик **O**. Если поле уже помечено, то попытка изменить его состояние будет проигнорирована. Также в этом свойстве осуществляется проверка выигрышной комбинации или ничьей с последующей обработкой этих событий. Обратите внимание, что в коде реализованы две перегруженные версии свойства. Вторая версия свойства позволяет указывать один параметр вместо двух: `iRow` и `iCol`. Проверка выигрышной или ничейной ситуации осуществляется просмотром состояния квадратов по горизонтали, вертикали или

диагонали. Если игрок имеет три квадрата в ряд в одном состоянии, значит, он выиграл. Обнаружение ничьей происходит через функцию `CheckDraw`: запускаем цикл по всем квадратам, если мы видим значение `Empty`, то возвращаем `False`; если все клетки заполнены, то значит — ничья.

Искусственный интеллект

Логика искусственного интеллекта содержится в классе `GameAI` (файл GameAI.vb). Есть четыре общих правила, которых придерживается AI (искусственный интеллект) противника (листинг 18.8):

- ❒ победить (если это возможно);
- ❒ помешать оппоненту выиграть;
- ❒ определить лучший ход для будущей победы;
- ❒ сделать следующий ход по правилам.

Листинг 18.8. Класс для искусственного интеллекта игры

```
'--------------------------------------
' Крестики-нолики 2 © 2004 Александр Климов
'--------------------------------------
Public Class GameAI
    Private m_iAIPlayer As GameModel.SquareStatus
    Private m_iHumanPlayer As GameModel.SquareStatus
    Private m_CurrentBoard As GameModel

    Public Sub New(ByVal iAIPlayer As GameModel.SquareStatus)
        m_iAIPlayer = iAIPlayer

        If iAIPlayer = GameModel.SquareStatus.PlayerX Then
            m_iHumanPlayer = GameModel.SquareStatus.PlayerO
        Else
            m_iHumanPlayer = GameModel.SquareStatus.PlayerX
        End If
    End Sub

    Public Function GetAIMove(ByVal aTTT As GameModel) As SquareCoordi-
nate
        Dim MoveList As New ArrayList
        m_CurrentBoard = aTTT
```

```
Dim iRow As Integer
For iRow = 0 To 2
    Dim possibleMoves() As SquareCoordinate
    Try
        possibleMoves = GetPossibleRowMoves(iRow)
    Catch ex As NullReferenceException
        Stop
    Catch ex As Exception
        Stop
    End Try

    If Not possibleMoves Is Nothing Then
        Dim x As Integer
        For x = 0 To possibleMoves.Length - 1
            MoveList.Add(DetermineMoveEntry(possibleMoves(x)))
        Next
    End If
Next

' Возвращает блокирующий ход или следующий обычный ход
Dim NormalMoveList As New ArrayList
Dim aMoveEntry As MoveEntry

' Проверка на выигрышный ход
For Each aMoveEntry In MoveList
    Select Case aMoveEntry.MoveType
        Case MoveType.Winning
            Console.Write(ControlChars.CrLf &
            aMoveEntry.ToString)
            Return aMoveEntry.SquareCoordinate
    End Select
Next

' Если нет победы, то блокирующий ход
For Each aMoveEntry In MoveList
    Select Case aMoveEntry.MoveType
        Case MoveType.Blocking
            Console.Write(ControlChars.CrLf &
            aMoveEntry.ToString)
```

```
                Return aMoveEntry.SquareCoordinate
            Case Else
                NormalMoveList.Add(aMoveEntry)
        End Select
    Next

    ' Поиск наилучшего следующего хода
    Try
        Dim bnME As SquareCoordinate = _
        DetermineBestNormalMove(NormalMoveList)
        Console.Write(ControlChars.CrLf & "Best -->" & bnME.ToString)
        Return bnME
    Catch ex As Exception

    End Try

    ' Случайный ход
    Dim iRandomItem As Integer = CInt(Int(((NormalMoveList.Count - 1)
    - 0 + 1) * Rnd() + 0))
    Dim nME As MoveEntry = NormalMoveList(iRandomItem)
    Console.Write(ControlChars.CrLf & nME.ToString)
    Return nME.SquareCoordinate
End Function

Private Function GetPossibleRowMoves(ByVal iRow As Integer) As
SquareCoordinate()
    Dim PossibleMoves As New ArrayList
    Dim iCol As Integer
    For iCol = 0 To 2
        If m_CurrentBoard.Square(iRow, iCol) =
        GameModel.SquareStatus.Empty Then
            PossibleMoves.Add(New SquareCoordinate(iRow, iCol))
        End If
    Next

    If PossibleMoves.Count > 0 Then
        Dim returnSC(PossibleMoves.Count - 1) As SquareCoordinate
        Dim x As Integer
        For x = 0 To PossibleMoves.Count - 1
            returnSC(x) = CType(PossibleMoves(x), SquareCoordinate)
```

```
        Next
        Return returnSC
    Else
        Dim returnSC() As SquareCoordinate
        Return returnSC
    End If
End Function

Private Function DetermineMoveEntry(ByVal aSC As SquareCoordinate) As
MoveEntry
    Dim iRow As Integer = aSC.Row
    Dim iCol As Integer = aSC.Column

    ' По горизонтали
    Dim iGE1 As GameModel.SquareStatus
    Dim iGE2 As GameModel.SquareStatus

    Select Case iCol
        Case 0
            iGE1 = m_CurrentBoard.Square(iRow, 1)
            iGE2 = m_CurrentBoard.Square(iRow, 2)
        Case 1
            iGE1 = m_CurrentBoard.Square(iRow, 0)
            iGE2 = m_CurrentBoard.Square(iRow, 2)
        Case 2
            iGE1 = m_CurrentBoard.Square(iRow, 0)
            iGE2 = m_CurrentBoard.Square(iRow, 1)
    End Select

    If CheckPossibleWin(iGE1, iGE2, m_iAIPlayer) = True Then
        Return New MoveEntry(aSC, MoveType.Winning)
    End If

    If CheckPossibleWin(iGE1, iGE2, m_iHumanPlayer) = True Then
        Return New MoveEntry(aSC, MoveType.Blocking)
    End If

    ' По вертикали
    Select Case iRow
```

```
        Case 0
            iGE1 = m_CurrentBoard.Square(1, iCol)
            iGE2 = m_CurrentBoard.Square(2, iCol)
        Case 1
            iGE1 = m_CurrentBoard.Square(0, iCol)
            iGE2 = m_CurrentBoard.Square(2, iCol)
        Case 2
            iGE1 = m_CurrentBoard.Square(0, iCol)
            iGE2 = m_CurrentBoard.Square(1, iCol)
    End Select

    If CheckPossibleWin(iGE1, iGE2, m_iAIPlayer) = True Then
        Return New MoveEntry(aSC, MoveType.Winning)
    End If

    If CheckPossibleWin(iGE1, iGE2, m_iHumanPlayer) = True Then
        Return New MoveEntry(aSC, MoveType.Blocking)
    End If

    ' По диагонали
    Select Case GetSquareType(aSC)
        Case SquareType.UpperLeft
            iGE1 = m_CurrentBoard.Square(1, 1)
            iGE2 = m_CurrentBoard.Square(2, 2)

            If CheckPossibleWin(iGE1, iGE2, m_iAIPlayer) = True Then
                Return New MoveEntry(aSC, MoveType.Winning)
            End If

            If CheckPossibleWin(iGE1, iGE2, m_iHumanPlayer) = True
            Then
                Return New MoveEntry(aSC, MoveType.Blocking)
            End If
        Case SquareType.UpperRight
            iGE1 = m_CurrentBoard.Square(1, 1)
            iGE2 = m_CurrentBoard.Square(2, 0)

            If CheckPossibleWin(iGE1, iGE2, m_iAIPlayer) = True Then
                Return New MoveEntry(aSC, MoveType.Winning)
            End If
```

```
        If CheckPossibleWin(iGE1, iGE2, m_iHumanPlayer) = True
        Then
            Return New MoveEntry(aSC, MoveType.Blocking)
        End If

    Case SquareType.Center
        iGE1 = m_CurrentBoard.Square(0, 2)
        iGE2 = m_CurrentBoard.Square(2, 0)

        If CheckPossibleWin(iGE1, iGE2, m_iAIPlayer) = True Then
            Return New MoveEntry(aSC, MoveType.Winning)
        End If

        If CheckPossibleWin(iGE1, iGE2, m_iHumanPlayer) = True
        Then
            Return New MoveEntry(aSC, MoveType.Blocking)
        End If

        iGE1 = m_CurrentBoard.Square(0, 0)
        iGE2 = m_CurrentBoard.Square(2, 2)

        If CheckPossibleWin(iGE1, iGE2, m_iAIPlayer) = True Then
            Return New MoveEntry(aSC, MoveType.Winning)
        End If

        If CheckPossibleWin(iGE1, iGE2, m_iHumanPlayer) = True
        Then
            Return New MoveEntry(aSC, MoveType.Blocking)
        End If
    Case SquareType.BottomLeft
        iGE1 = m_CurrentBoard.Square(1, 1)
        iGE2 = m_CurrentBoard.Square(0, 2)

        If CheckPossibleWin(iGE1, iGE2, m_iAIPlayer) = True Then
            Return New MoveEntry(aSC, MoveType.Winning)

        End If
        If CheckPossibleWin(iGE1, iGE2, m_iHumanPlayer) = True
        Then
            Return New MoveEntry(aSC, MoveType.Blocking)
```

```
                End If
            Case SquareType.BottomRight
                iGE1 = m_CurrentBoard.Square(1, 1)
                iGE2 = m_CurrentBoard.Square(0, 0)

                If CheckPossibleWin(iGE1, iGE2, m_iAIPlayer) = True Then
                    Return New MoveEntry(aSC, MoveType.Winning)
                End If

                If CheckPossibleWin(iGE1, iGE2, m_iHumanPlayer) = True
                Then
                    Return New MoveEntry(aSC, MoveType.Blocking)
                End If
        End Select

        Return New MoveEntry(aSC, MoveType.Normal)
    End Function

    Private Function DetermineBestNormalMove(ByVal aNormalMoveList As
ArrayList) As SquareCoordinate
        Dim aSC As SquareCoordinate
        Dim aME As MoveEntry
        Dim aGE1 As GameModel.SquareStatus
        Dim aGE2 As GameModel.SquareStatus


        For Each aME In aNormalMoveList
            aSC = aME.SquareCoordinate
            ' По горизонтали
            If CheckBestHorizontal(aSC) Then
                Return aSC
            End If

            If CheckBestVertical(aSC) Then
                Return aSC
            End If

            If GetSquareType(aSC) <> SquareType.Other Then
                If CheckBestDiagonal(aSC) Then
```

```
                    Return aSC
                End If
            End If
        Next
    End Function

    Private Function CheckBestHorizontal(ByVal aSC As SquareCoordinate)
As Boolean
        Dim iCol As Integer = aSC.Column
        Dim iRow As Integer = aSC.Row
        Dim iGE1 As GameModel.SquareStatus
        Dim iGE2 As GameModel.SquareStatus

        Select Case iCol
            Case 0
                iGE1 = m_CurrentBoard.Square(iRow, 1)
                iGE2 = m_CurrentBoard.Square(iRow, 2)
            Case 1
                iGE1 = m_CurrentBoard.Square(iRow, 0)
                iGE2 = m_CurrentBoard.Square(iRow, 2)
            Case 2
                iGE1 = m_CurrentBoard.Square(iRow, 0)
                iGE2 = m_CurrentBoard.Square(iRow, 1)
        End Select

        If iGE1 = m_iHumanPlayer Or iGE2 = m_iHumanPlayer Then
            Return False
        End If

        If iGE1 = m_iAIPlayer Or iGE2 = m_iAIPlayer Then
            Return True
        End If

        Return False
    End Function

    Private Function CheckBestVertical(ByVal aSC As SquareCoordinate) As
Boolean
        Dim iCol As Integer = aSC.Column
```

```
        Dim iRow As Integer = aSC.Row
        Dim iGE1 As GameModel.SquareStatus
        Dim iGE2 As GameModel.SquareStatus

        Select Case iRow
            Case 0
                iGE1 = m_CurrentBoard.Square(1, iCol)
                iGE2 = m_CurrentBoard.Square(2, iCol)
            Case 1
                iGE1 = m_CurrentBoard.Square(0, iCol)
                iGE2 = m_CurrentBoard.Square(2, iCol)
            Case 2
                iGE1 = m_CurrentBoard.Square(0, iCol)
                iGE2 = m_CurrentBoard.Square(1, iCol)
        End Select

        If iGE1 = m_iHumanPlayer Or iGE2 = m_iHumanPlayer Then
            Return False
        End If

        If iGE1 = m_iAIPlayer Or iGE2 = m_iAIPlayer Then
            Return True
        End If

        Return False
    End Function

    Private Function CheckBestDiagonal(ByVal aSC As SquareCoordinate) As
Boolean
        Return False
    End Function

    Private Enum SquareType
        UpperLeft = 1
        UpperRight = 2
        Center = 3
        BottomLeft = 4
        BottomRight = 5
        Other = 6
    End Enum
```

```
    Private Function CheckPossibleWin(ByVal iGE1 As
    GameModel.SquareStatus, _
                                      ByVal iGE2 As
    GameModel.SquareStatus, _
                                      ByVal iPlayer As
    GameModel.SquareStatus) As Boolean


        If (iGE1 = iPlayer) And (iGE2 = iPlayer) Then
            Return True
        Else
            Return False
        End If
    End Function


    Private Function GetSquareType(ByVal aSC As SquareCoordinate) As
SquareType
        If aSC.Row = 0 And aSC.Column = 0 Then Return
        SquareType.UpperLeft
        If aSC.Row = 0 And aSC.Column = 2 Then Return
        SquareType.UpperRight
        If aSC.Row = 1 And aSC.Column = 1 Then Return SquareType.Center
        If aSC.Row = 2 And aSC.Column = 0 Then Return
        SquareType.BottomLeft
        If aSC.Row = 2 And aSC.Column = 2 Then Return
        SquareType.BottomRight
        Return SquareType.Other
    End Function


    Enum MoveType
        Normal = 0
        Blocking = 1
        Winning = 2
    End Enum


    Private Class MoveEntry
        Private m_iMoveType As MoveType
        Private m_SquareCoordinate As SquareCoordinate


        Public Sub New(ByVal aSC As SquareCoordinate, ByVal iMoveType As
MoveType)
            m_iMoveType = iMoveType
```

```
            m_SquareCoordinate = aSC
        End Sub

        Public ReadOnly Property [MoveType]() As MoveType
            Get
                Return m_iMoveType
            End Get
        End Property

        Public ReadOnly Property [SquareCoordinate]() As SquareCoordinate
            Get
                Return m_SquareCoordinate
            End Get
        End Property

        Public Overrides Function ToString() As String
            Dim sOutput As String
            sOutput = Me.MoveType.ToString & " : " & _
                      "[" & Me.m_SquareCoordinate.Row & ", " & _
                     Me.m_SquareCoordinate.Column & "]"
            Return sOutput
        End Function
    End Class
End Class
```

Итак, мы выстроили модель игры и искусственного интеллекта. Теперь можно писать код для самой игры (листинг 18.9).

Листинг 18.9. Код для игры "Крестики-нолики 2"

```
'--------------------------------------
' Крестики-нолики 2 © 2005 Александр Климов
'--------------------------------------
    Private m_TicTacToeGame As New GameModel
    Private m_CurPlayer As GameModel.SquareStatus =
    GameModel.SquareStatus.PlayerX
    Private m_TicTacToeAI As New GameAI(GameModel.SquareStatus.PlayerO)
    Private m_bIsAIEnabled As Boolean = True
    Private m_bGameEnded As Boolean = False
```

```
    Private Sub lbl00_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles lbl00.Click, lbl01.Click, lbl02.Click,
lbl10.Click, lbl11.Click, lbl12.Click, lbl20.Click, lbl21.Click,
lbl22.Click
        Try
            Dim aLabelControl As Label = CType(sender, Label)
            Dim sName As String = aLabelControl.Name
            Dim iRow As Integer = sName.Substring(sName.Length - 2, 1)
            Dim iCol As Integer = sName.Substring(sName.Length - 1, 1)
            If m_TicTacToeGame.Square(iRow, iCol) =
            GameModel.SquareStatus.Empty Then
                m_TicTacToeGame.Square(iRow, iCol) = m_CurPlayer
                ToggleCurrentPlayer()
                DrawBoard()
            End If
        Catch ex As Exception
            Stop
        End Try

    End Sub
    Private Sub ToggleCurrentPlayer()
        If m_CurPlayer = GameModel.SquareStatus.PlayerX Then
            m_CurPlayer = GameModel.SquareStatus.PlayerO
            If m_bGameEnded = False Then
                PerformAIMove()
            End If
        Else
            m_CurPlayer = GameModel.SquareStatus.PlayerX
        End If
    End Sub
    Private Sub DrawBoard()
        Dim iRow As Integer
        Dim iCol As Integer
        Dim aLabelControl As Label

        For iRow = 0 To 2
            For iCol = 0 To 2
                Try
                    aLabelControl = GetLabelControl(iRow, iCol)
                    Select Case m_TicTacToeGame.Square(iRow, iCol)
```

```
                    Case GameModel.SquareStatus.Empty
                        aLabelControl.Text = ""
                    Case GameModel.SquareStatus.PlayerX
                        aLabelControl.Text = "X"
                    Case GameModel.SquareStatus.PlayerO
                        aLabelControl.Text = "O"
                End Select
            Catch ex As Exception

            End Try
        Next
    Next

End Sub

Private Sub PerformAIMove()
    Try
        Dim aSC As SquareCoordinate
        Try
            aSC = _
            m_TicTacToeAI.GetAIMove(m_TicTacToeGame)

        Catch ex As NullReferenceException
            Stop
        Catch ex As Exception
            Stop
        End Try
        m_TicTacToeGame.Square(aSC) = GameModel.SquareStatus.PlayerO
        DrawBoard()
        ToggleCurrentPlayer()
    Catch ex As Exception
        Stop
    End Try

End Sub

    Private Function GetLabelControl(ByVal iRow As Integer, ByVal iCol As
Integer) As Label
        Dim x As Integer
```

```
    Dim aControl As Control
    For x = 0 To Me.Controls.Count - 1
        aControl = Me.Controls(x)
        If aControl.Name = "lbl" & iRow & iCol Then Return aControl
    Next
End Function

Private Sub DrawState()
    DrawBoard()
    MessageBox.Show("Ничья.")
    m_bGameEnded = True
    NewGame()
End Sub

Private Sub NewGame()
    Dim rtnDlgResult As DialogResult
    rtnDlgResult = MessageBox.Show("Вы хотите сыграть снова?",
    "Новая игра", MessageBoxButtons.YesNo)
    If rtnDlgResult = DialogResult.Yes Then
        m_TicTacToeGame.Clear()
        m_bGameEnded = False
    Else
        Me.Close()
    End If
End Sub
Private Sub WeHaveAWinner(ByVal iWinner As GameModel.SquareStatus)
    DrawBoard()
    Dim sMessage As String = "Игрок "
    Select Case iWinner
        Case GameModel.SquareStatus.PlayerX
            sMessage = sMessage & " X выиграл."
        Case GameModel.SquareStatus.PlayerO
            sMessage = sMessage & " O выиграл."
    End Select
    MessageBox.Show(sMessage)
    m_bGameEnded = True
    NewGame()
End Sub
```

```
    Private Sub ReSet()
        m_TicTacToeGame.Clear()
        DrawBoard()
    End Sub
```

Приведенный пример практически без изменений использует код, выложенный на сайте **http://www.codeproject.com**. На этом сайте можно найти еще несколько вариантов данной игры, реализованных на разных языках программирования.

18.4. Карточные игры

Уверен, что каждый из вас играл в карточные игры, входящие в состав Windows: "Солитер", "Черви", "Косынка", "Паук". Это достаточно простые игры, в которые вы играли много раз. Но может быть вам хотелось поиграть в более сложные игры? В таком случае напишите игру сами! Дело упрощается тем, что вышеупомянутые игры используют библиотеку cards.dll, которая поставляется с Windows XP. Опираясь на эту библиотеку, вы способны создать любую карточную игру на Visual Basic .NET 2003 по собственному вкусу.

Примечание

Библиотека cards.dll входит в состав практически любой версии Windows. Но в каждой Windows использовались разные версии библиотеки, как 16-битные, так и 32-битные. Кроме того, иногда файлы этой библиотеки имели разные имена, например, cards32.dll. В нашем случае мы рассматриваем только библиотеку, поставляемую вместе с Windows XP.

Данная библиотека содержит три основные функции, необходимые для создания карточных игр:

```
Declare Function cdtInit Lib "cards.dll" (ByRef width As Integer, _
  ByRef height As Integer) As Boolean
Declare Function cdtDrawExt Lib "cards.dll" (ByVal hdc As IntPtr, _
  ByVal x As Integer, ByVal y As Integer, ByVal dx As Integer, _
  ByVal dy As Integer, ByVal card As Integer, _
  ByVal suit As Integer, ByVal color As Long) As Boolean
Declare Sub cdtTerm Lib "cards.dll" ()
```

Функция `cdtInit` применяется для инициализации библиотеки cards.dll. В переменных `width` и `height` содержатся размеры карт, необходимые при выводе их на экран. Функция `cdtTerm` служит для завершения работы с библиотекой cards.dll и освобождения ресурсов памяти. Основная функция,

используемая для создания игры, — функция `cdtDrawExt`. Она служит для вывода карт на экран. Приведем параметры, используемые функцией:

- `hdc` — контекст устройства формы или `PictureBox`;
- `x` — координата `X` левой стороны карты в пикселах;
- `y` — координата `Y` верхней стороны карты в пикселах;
- `dx` — ширина карты;
- `dy` — высота карты;
- `card` — число от 0 до 55, описывающее тип выводимой карты;
- `suit` — значение, описывающее метод вывода карты;
- `color` — инвертируемый цвет.

Теперь, когда мы познакомились с основными функциями, можно приступать к созданию игры. Сначала рассмотрим простейший пример вывода на форму одной карты. Надеюсь, вы уже запустили новый проект и приготовились писать код? Итак, сначала объявим две глобальные переменные `w` и `h`, в которых будут содержаться размеры карт (листинг 18.10).

Листинг 18.10. Создание карточной игры

```
'--------------------------------------
' Создание карточной игры © 2004 Александр Климов
'--------------------------------------
    Private w As Integer = 0
    Private h As Integer = 0

    ' Инициализируем карточную библиотеку
    Private Sub Form1_Load(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles MyBase.Load
        cdtInit(w, h)
    End Sub

    Private Sub Form1_Paint(ByVal sender As Object, _
      ByVal e As System.Windows.Forms.PaintEventArgs) Handles
      MyBase.Paint
        DrawOneCard(e.Graphics, 10, 10, w, h)
    End Sub

    Declare Function cdtInit Lib "cards.dll" (ByRef width As Integer, _
      ByRef height As Integer) As Boolean
```

```
Declare Function cdtDrawExt Lib "cards.dll" (ByVal hdc As IntPtr, _
  ByVal x As Integer, ByVal y As Integer, ByVal dx As Integer, _
  ByVal dy As Integer, ByVal card As Integer, _
  ByVal suit As Integer, ByVal color As Long) As Boolean
Declare Sub cdtTerm Lib "cards.dll" ()


Public Sub DrawOneCard(ByVal g As Graphics, ByVal x As Integer, _
      ByVal y As Integer, ByVal dx As Integer, ByVal dy As Integer)
    Dim FCardFace As Integer = 1
    Dim FSuit As Integer = 2
    Dim card As Integer
    Dim hdc As IntPtr = g.GetHdc()
    Try
        card = CType(FCardFace, Integer) * 4 + FSuit
        cdtDrawExt(hdc, x, y, dx, dy, 0, 0, 0)
    Finally
        g.ReleaseHdc(hdc)
    End Try
End Sub

Private Sub Form1_Closing(ByVal sender As Object, _
  ByVal e As System.ComponentModel.CancelEventArgs) Handles
  MyBase.Closing
    cdtTerm()
End Sub
```

Теперь, после инициализации, в переменных `w` и `h` содержатся реальные размеры карт. Можете проверить, вставив после инициализации две строчки:

```
' Узнаем размеры карт (для проверки)
Debug.WriteLine(w)
Debug.WriteLine(h)
```

Эти величины использовать в вашей игре не обязательно, вы можете устанавливать собственные размеры для карт. Теперь приступим непосредственно к выводу карты на экран. Для первого знакомства я вынес процедуру вывода одной предопределенной карты в отдельную процедуру `DrawOneCard`. Эта процедура очень проста. Сначала мы получаем дескриптор контекста устройства, на котором будем выводить изображение карты, а затем вызываем функцию `cdtDrawExt` с необходимыми параметрами.

Об этой функции стоит рассказать поподробнее. Как вы знаете, стандартная колода состоит из 52 карт. Карты имеют четыре масти по 13 наименований в каждой. Параметру `card` соответствует число, определяющее достоинство карты. Числа в интервале `0`–`51` выводят одну из стандартных 52 карт колоды. Так, значение `0` соответствует тузу треф (♣), `1` — тузу бубен (♦), `2` — тузу червей (♥), `3` — тузу пик (♠) и т. д. Последнее число `51` соответствует королю пик. Остальные числа от `52` до `55` выводят различные виды рубашек, а также несколько специальных картинок (**X** или **O**). Перечеркнутая красным крестом карта обозначает конец игры, а зеленая **O** указывает, что вы можете перемешать колоду для продолжения игры. Параметр `suit` позволяет выводить различные состояния карты. Например, вы присвоили параметру `card` значение от `0` до `51` (нормальный вид). В этом случае при значении `0` в параметре `suit` вы выводите стандартный вид карты. Передавая значение `2`, вы выводите эту же карту в инвертированном виде, показывая пользователю, что карта выбрана. Если же необходимо показать рубашку карты (обратную сторону), то используйте значение `1` в параметре `suit` при значениях `card` в интервале от `53` до `68`. Последний параметр `color` функцией не используется, поэтому установите его равным `0`. Поместим вызов описанной процедуры `DrawOneCard` в обработчик события `Form1_Paint` и запустим проект:

```
Private Sub Form1_Paint(ByVal sender As Object, _
  ByVal e As System.Windows.Forms.PaintEventArgs) Handles
  MyBase.Paint
    DrawOneCard(e.Graphics, 10, 10, w, h)
End Sub
```

В левом углу формы у вас должно появиться изображение трефового туза (рис. 18.5). При выходе из программы необходимо вызвать процедуру `cdtTerm` для освобождения ресурсов:

```
Private Sub Form1_Closing(ByVal sender As Object, _
  ByVal e As System.ComponentModel.CancelEventArgs) Handles
  MyBase.Closing
    cdtTerm()
End Sub
```

Приведенной выше информации вполне достаточно, чтобы вы уже начали создавать свою карточную игру. Если вы думаете, что я это сделаю за вас, то глубоко ошибаетесь! Впрочем, заглядывайте иногда на мой сайт **http://rusproject.narod.ru**, возможно, я выложу туда какую-нибудь реализацию игры.

Рис. 18.5. Вывод карты на экран

18.5. "Змейка"

"Змейка" — одна из самых распространенных компьютерных игр. Она появилась еще на первых электронных машинах, так как для ее реализации не требовались графические возможности. Известна она любителям игр под разными названиями: "Змейка", "Удав" и т. п. Игра выглядит следующим образом. По полю передвигается маленькая змейка, управляемая клавишами. На поле в случайном порядке появляется для нее пища — условно ею является кролик. Цель игры — съесть предлагаемую пищу. При каждом съеденном кролике длина змеи увеличивается. У игры есть одно ограничение — змея не должна есть сама себя. Если голова змеи встретится с ее туловищем, то игра прекращается. В некоторых играх также нельзя касаться границ поля. В начале игра кажется простой. Но с увеличением размеров змеи (и скорости ее передвижения) задача становится все более сложной.

По счастливому стечению обстоятельств, когда я только приступил к написанию собственной версии игры, мне вдруг попался на глаза интересный сайт, посвященный этой игре. Причем на сайте рассматривались реализации игры на разных языках программирования. Код программы на VB .NET расположен на странице **http://neuron.et.ntust.edu.tw/homework/91/PL/A9015047/VB.NETSnakeGame.htm**. Я внес в программу небольшие косметические изменения, но она настолько хорошо написана, что мне пришлось отказаться от дальнейшей модификации. Хочу предложить вам самостоятельно разобрать эту игру "по косточкам". По возможности я добавил в код небольшие комментарии, которые помогут вам понять принцип игры. Код игры большой и сложный, состоящий из нескольких классов. Для экономии места я не стану приводить в книге весь код, который вы можете найти на прилагаемом диске. Приведу только небольшой отрывок. В новом проекте

Snake поместите на форму элемент PictureBox, который будет служить поляной для змейки. Установите для формы Form1 следующие свойства:

- Size = 472, 405;
- Text = Змейка.

Для PictureBox1 установите следующие свойства:

- Anchor = Top, Bottom, Left, Right;
- BackColor = DarkOrange;
- BorderStyle = Fixed3D;
- Size = 450, 350.

Еще нам понадобится несколько надписей Label и таймер Timer. Поместите на форму надписи в любом удобном вам месте. В них должна содержаться информация, которая поможет игроку разобраться с управлением игрой. Я постарался разместить надписи приблизительно в центре формы. Цвет фона и текста установите по вашему желанию. На рис. 18.6 вы можете увидеть мой вариант начального экрана игры. Визуальное построение игры закончено (листинг 18.11).

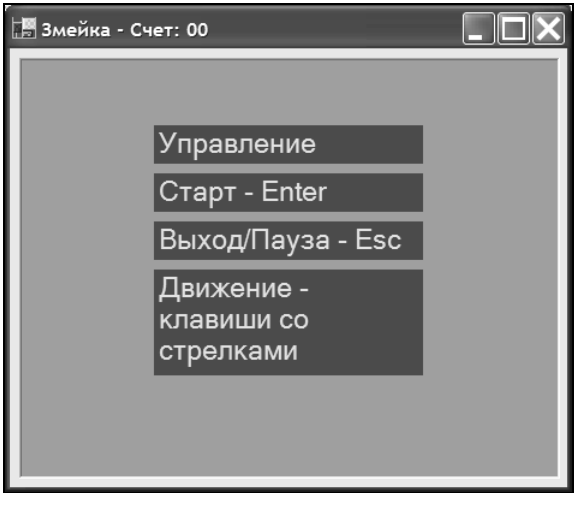

Рис. 18.6. Начальный экран игры

Листинг 18.11. Игра "Змейка"

```
'--------------------------------------
' Игра Змейка © 2004 Александр Климов
'--------------------------------------
```

```
' Form1.vb
    ' Сколько сегментов добавляем змейке
    Private Const GROWTH_FACTOR As Integer = 3

    ' Ширина сегмента в пикселах
    Private Const SNAKE_WIDTH As Integer = 8

    ' Объект змейка
    Private m_snake As snake

    ' Контроль над движением змейки
    Private m_control As SnakeControl

    ' Идет ли игра?
    Private m_running As Boolean = False

    ' Увеличивается ли размер змейки
    Private m_growing As Boolean = False

    ' Прямоугольник, использующийся для пищи (размеры и позиция)
    Private m_foodrec As Rectangle

    ' Счет игры
    Private m_score As Integer

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        ' Начинаем игру
        StartGame()
    End Sub

    Private Sub PictureBox1_Paint(ByVal sender As Object, ByVal e As
System.Windows.Forms.PaintEventArgs) Handles PictureBox1.Paint
        ' Если игра идет
        If Not m_running Then

            e.Graphics.Clear(PictureBox1.BackColor)

            Exit Sub

        End If
```

```
        ' Рисуем пищу
        e.Graphics.FillEllipse(Brushes.Cornsilk, m_foodrec)

        ' Временная переменная для работы с сегментами
        ' змейки в цикле
        Dim thisseg As SnakeSegment

        ' Проходим через все сегменты и рисуем их
        For Each thisseg In m_snake.Segments

            ' Каждый сегмент является небольшим цветным прямоугольником
            e.Graphics.FillRectangle(_
          Brushes.LimeGreen, thisseg.Rectangle)
        Next

    End Sub

    Private Sub Timer1_Tick(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Timer1.Tick
        ' Обновляем игру
        UpdateGame()

        ' Перерисовываем игровое поле
        PictureBox1.Invalidate()

    End Sub

    Private Sub GameOver()

        ShowMsg()
        MsgBox("Игра окончена!", vbOKOnly, "Змейка")
        Label1.Visible = True
        Label2.Visible = True
        Label3.Visible = True
        Label4.Visible = True

        ' Запускаем игру сначала
        StartGame()

    End Sub
```

```
    Public Sub PlaceFood()
        ' Находим случайную позицию на поляне
        ' для пищи змеи

        Dim foodpt As Point

        ' Следим за тем, чтобы пища не оказалась
        ' на туловище змейки
        Do
            foodpt = GetRandomPoint()

            ' Есть ли змейка
            If Not (m_snake Is Nothing) Then

                ' Проверяем, находится ли пища на змейке
                If Not m_snake.PointOnSnake(foodpt) Then Exit Do
            Else
                Exit Do
            End If

        Loop
        ' Находим позицию для пищи
        m_foodrec.Location = foodpt

    End Sub

    Private Sub frmMain_KeyUp(ByVal sender As Object, ByVal e As
System.Windows.Forms.KeyEventArgs) Handles MyBase.KeyUp
        Select Case e.KeyCode

            Case Keys.Enter ' Начало игры
                HideMsg()
                Cursor.Hide()
            Case Keys.Escape ' Пауза/Выход
                If m_running Then
                    ShowMsg()
                Else
                    Me.Close()
                    Cursor.Show()
```

```
            End If
        End Select

    End Sub

    Private Sub frmMain_KeyDown(ByVal sender As Object, ByVal e As
System.Windows.Forms.KeyEventArgs) Handles MyBase.KeyDown
        Select Case e.KeyCode

            Case Keys.Right, Keys.NumPad6 ' движение направо

                m_control.Direction = SnakeControl.SnakeDirection.Right

            Case Keys.Down, Keys.NumPad2 ' вниз

                m_control.Direction = SnakeControl.SnakeDirection.Down

            Case Keys.Left, Keys.NumPad4 ' влево

                m_control.Direction = SnakeControl.SnakeDirection.Left

            Case Keys.Up, Keys.NumPad8 ' вверх

                m_control.Direction = SnakeControl.SnakeDirection.Up

            End Select

    End Sub

    Private Sub StartGame()
        ' Начинаем игру

        ' Обновляем счетчик
        m_score = 0

        ' Инициализация прямоугольника для пищи
        m_foodrec = New Rectangle(0, 0, SNAKE_WIDTH, SNAKE_WIDTH)

        ' Размещаем пищу на поляне
        PlaceFood()
```

```
        ' Начальная позиция змейки (середина графического поля)
        Dim startpt As New Point( _
        CInt(PictureBox1.ClientSize.Width / 2 / SNAKE_WIDTH + 0.5) *
        SNAKE_WIDTH, _
        CInt(PictureBox1.ClientSize.Height / 2 / SNAKE_WIDTH + 0.5) *
        SNAKE_WIDTH)

        ' Инициализация змейки с одним сегментом
        m_snake = New snake(startpt, SNAKE_WIDTH, 1)

        ' Инициализация управления змейки
        m_control = New SnakeControl(SNAKE_WIDTH, m_snake.Head.Location,
SnakeControl.SnakeDirection.Right)

        ' В начале игры идет небольшое увеличение размеров змейки
        m_growing = True

    End Sub

    Private Sub UpdateGame()

        ' Число сегментов для увеличения размеров змейки
        Static targetgrowth As Integer = GROWTH_FACTOR

        ' Число уже добавленных сегментов
        Static segmentsadded As Integer

        ' если игра окончена, то выходим
        If Not m_running Then Exit Sub

        ' включаем управление змейкой
        m_control.Move(PictureBox1.ClientRectangle)

        ' Если змейка укусила сама себя
        If m_snake.PointOnSnake(m_control.Location) Then
            ' сбрасываем значения переменных
            targetgrowth = 0
            segmentsadded = 0
```

```
        ' Игра окончена
        GameOver()
        Return

        ' Если змейка съела пищу
    ElseIf m_foodrec.Contains(m_control.Location) Then

        ' То она увеличивается в размерах
        targetgrowth += GROWTH_FACTOR
        m_growing = True

        ' Выдаем новую порцию еды
        PlaceFood()

        ' и увеличиваем счет
        m_score += 10
        Text = "Змейка - Счет: " + m_score.ToString
    End If

    ' Проверяем возможность роста у змейки
    If m_growing Then

        ' в этом случае сбрасываем значение
        If targetgrowth < GROWTH_FACTOR Then targetgrowth =
        GROWTH_FACTOR

        ' Если змейка достаточно большая
        If segmentsadded >= targetgrowth Then

            m_growing = False
            segmentsadded = 0
            targetgrowth = 0

            ' Змейка больше не растет,
            ' а просто двигается
            m_snake.Slither(m_control.Location)
```

```
        Else

            ' Добавляем новый сегмент в направлении движения змейки
            m_snake.Add(m_control.Location)

            ' Увеличиваем число добавленных сегментов
            segmentsadded += 1

        End If

    Else
        ' Движение змейки в новую точку
        m_snake.Slither(m_control.Location)
    End If

End Sub


Private Sub ShowMsg()
    ' Процедура для паузы в игре
    ' и вывода сообщения

    m_running = False
    Timer1.Enabled = False
    Label1.Visible = True
    Label2.Visible = True
    Label3.Visible = True
    Label4.Visible = True
End Sub

.
Private Sub HideMsg()
    ' Прячем предыдущее сообщение
    ' и продолжаем игру

    Label1.Visible = False
    Label2.Visible = False
```

```
        Label3.Visible = False
        Label4.Visible = False

        m_running = True

        Timer1.Enabled = True

    End Sub
```

И так далее. Запустите игру, нажмите клавишу <Enter> и начинайте управлять змейкой с помощью клавиш со стрелками (рис. 18.7).

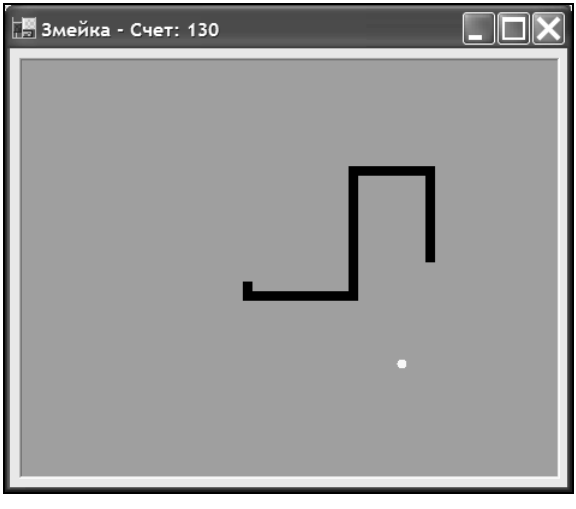

Рис. 18.7. Игра "Змейка"

18.6. Советы и рекомендации

Попробуйте улучшить игру "Сейф-2" (*см. разд. 18.2*). Например, вместо окрашивания кнопки другим цветом можно загрузить кусочек какой-нибудь картинки. Тогда у игрока будет дополнительный стимул, чтобы рассмотреть всю картинку.

Не очень удобно пользоваться числовыми данными при создании карточной игры. Попробуйте придумать способ, позволяющий вам сопоставить значение карты ее описанию. Например, для карты с значением `0` можно сопоставить переменную `ClubsAce`.

Глава 19

Трюки

В этой главе собраны некоторые трюки, к которым весьма часто прибегают программисты в своих программах. Причем эти же приемы активно используются в самых разных языках программирования. Надо сказать, что большинство из них основано на системных функциях Windows API. Невозможно в одной книге описать все функции, число которых исчисляется десятками тысяч. В помощь программистам я подготовил "Справочник по функциям Windows API в среде Visual Basic", где даны описания наиболее распространенных функций с примерами применительно как к VB 6.0, так и к VB .NET 2003. На моем сайте вы можете скачать демо-версию справочника, там же вы найдете информацию, как его приобрести.

19.1. Размещение окна программы в правом нижнем углу монитора

Иногда требуется аккуратно поместить окно своей программы в определенном месте с учетом размеров экрана и панели задач. Для решения этой задачи очень хорошо подходит свойство `WorkingArea` класса `SystemInformation`. Не забывайте, что у пользователя панель задач может располагаться не только внизу, но и сбоку или сверху экрана. Профессионально написанная программа должна учитывать эти обстоятельства и подстраиваться под вкусы пользователя. Можно поместить код расположения формы на экране прямо в событии `Load` проекта `RightCorner` (листинг 19.1).

Листинг 19.1. Размещение формы в заданном месте

```
'--------------------------------------
' Размещение формы в заданном месте © 2004 Александр Климов
'--------------------------------------

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
```

```
        Dim wa As Rectangle = SystemInformation.WorkingArea
        Dim x As Integer = wa.Left + wa.Width - Me.Width
        Dim y As Integer = wa.Top + wa.Height - Me.Height
        Me.Location = New Point(x, y)
    End Sub
```

При загрузке формы вычисляются размеры рабочей области экрана и путем нехитрых математических расчетов определяются координаты верхнего левого угла формы. Запустив проект, вы увидите, что окно программы расположилось точно над панелью задач (если она находится внизу) в правом нижнем углу экрана. Закройте проект и перетащите панель задач в другое место. Запустите проект снова и убедитесь, что программа по-прежнему выводится в правом нижнем углу.

19.2. Пасхальное яйцо

Иногда в целях защиты интеллектуальной собственности программисты прячут в своих программах скрытые элементы — диалоговые окна со списком разработчиков программы, картинки, зашифрованные послания, секретные команды. Подобного рода секреты называют *пасхальными яйцами* (easter egg). Действия, вызываемые пасхальными яйцами, зависят исключительно от фантазии разработчика. Порой, чтобы добраться до пасхального яйца и, например, поиграть на секретном уровне, приходится выполнить ряд сложных манипуляций с клавиатурой и мышью в определенной последовательности. Существуют целые сайты с подборкой известных пасхальных яиц в различных программах. Пасхальные яйца находили в таких популярных продуктах, как Internet Explorer, Windows 95, Adobe Photoshop, Microsoft Excel. Простейший пример пасхального яйца можно увидеть в популярном архиваторе WinRAR (**http:// http://www.rarlab.com/**). Запустите программу WinRAR, выберите команду **О программе**. У вас откроется окно с номером версии и именами разработчиков. Слева вы увидите логотип программы — стопку связанных книг. Щелкните мышью на этой стопке. И о, чудо! Стопка книг начнет медленно падать вниз, затем отскочит как резиновый мячик и подпрыгнет несколько раз.

Мы с вами рассмотрим один из способов создания пасхального яйца. Создаем новый проект `EasterEgg`. Пишем код, который будет отслеживать нажатия клавиатуры (листинг 19.2).

Листинг 19.2. Создание пасхального яйца

```
'--------------------------------------
' Пасхальное яйцо © 2004 Александр Климов
'--------------------------------------
```

```
    Private Sub Form1_KeyDown(ByVal sender As Object, ByVal e As
System.Windows.Forms.KeyEventArgs) Handles MyBase.KeyDown
        Const EASTER_EGG As String = "CAT"
        Static COMPARE_STRING As String

        COMPARE_STRING = COMPARE_STRING & e.KeyCode.ToString
        If Microsoft.VisualBasic.Left(EASTER_EGG, Len(COMPARE_STRING)) <>
COMPARE_STRING Then
            ' Если не совпадает, то начинаем снова
            COMPARE_STRING = ""
        Else
            ' Если совпало, то мяукаем
            If EASTER_EGG = COMPARE_STRING Then
                MsgBox("Мяу")
                COMPARE_STRING = ""
            End If
        End If
    End Sub
```

Теперь достаточно установить свойство формы `KeyPreview` в `True` и запустить пример. Наберите на клавиатуре слово `CAT`, и у вас выскочит соответствующее сообщение. Неплохо бы еще проиграть звуковой файл, воспроизводящий мяуканье кошки.

19.3. Запуск другого приложения

Если вам необходимо запустить другое приложение из своей программы, воспользуйтесь методом `Start` класса `Process`. Вот как можно запустить стандартный калькулятор, входящий в состав Windows — проект `RunApp` (листинг 19.3).

Листинг 19.3. Запуск другого приложения

```
'--------------------------------------
' Запуск других программ © 2004 Александр Климов
'--------------------------------------
   Private Sub butCalc_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles butCalc.Click
        ' запускаем калькулятор
        Process.Start("calc.exe")
    End Sub
```

Это самый простой пример. Можно запускать приложение с аргументами. Предположим, вам нужно запустить браузер Internet Explorer с открытием заданной страницы (листинг 19.4).

Листинг 19.4. Запуск заданной страницы браузером Internet Explorer

```
    Private Sub butIE_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles butIE.Click
        ' запускаем браузер с открытием заданной страницы
        Process.Start("IExplore.exe", "E:\mysite\vb\index.htm")
    End Sub
```

Возможно вам захочется не только открыть новое приложение, но и указать его местоположение и размеры его окна. В этом случае можно использовать функцию Windows API `MoveWindow`:

```
Private Declare Function MoveWindow Lib "user32" Alias "MoveWindow" _
        (ByVal hwnd As IntPtr, ByVal x As Integer, _
         ByVal y As Integer, ByVal nWidth As Integer, _
         ByVal nHeight As Integer, _
         ByVal bRepaint As Integer) As Integer
```

И после вызова приложения передать этой функции требуемые параметры. Вот как запускается стандартный Блокнот с заданными размерами (листинг 19.5).

Листинг 19.5. Запуск Блокнота в заданном месте и с заданными размерами

```
    Private Sub butNotepad_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles butNotepad.Click
        ' запускаем блокнот
        Dim p As System.Diagnostics.Process = _
           System.Diagnostics.Process.Start("notepad.exe")
        p.WaitForInputIdle()
        ' перемещаем и устанавливаем новые размеры окна блокнота
        MoveWindow(p.MainWindowHandle, 0, 10, 700, 200, 1)
    End Sub
```

19.4. Запрет на запуск второй копии приложения

Иногда требуется запретить пользователю запускать более одного экземпляра программы. Самый простой пример — почтовая программа Outlook

Express. Вам не удастся запустить несколько копий этой программы, что вполне справедливо. Иначе возникла бы путаница с получением и отправкой писем. Существует несколько методик, не позволяющих запускать второй экземпляр запущенной программы. Я хочу предложить вам один из вариантов — проект `OneInstance` (листинг 19.6).

Листинг 19.6. Запрет на запуск второй копии программы

```
'--------------------------------------
' Запрет на запуск второй копии программы © 2004 Александр Климов
'--------------------------------------
    Public Enum ShowWindowConstants
        SW_HIDE = 0
        SW_SHOWNORMAL = 1
        SW_SHOWMINIMIZED = 2
        SW_SHOWMAXIMIZED = 3
        SW_MAXIMIZE = 3
        SW_SHOWNOACTIVATE = 4
        SW_SHOW = 5
        SW_MINIMIZE = 6
        SW_SHOWMINNOACTIVE = 7
        SW_SHOWNA = 8
        SW_RESTORE = 9
        SW_SHOWDEFAULT = 10
        SW_FORCEMINIMIZE = 11
    End Enum

    Declare Auto Function ShowWindowAsync Lib "user32.dll" ( _
        ByVal hwnd As IntPtr, _
        ByVal nCmdShow As Int32 _
      ) As Int32
    Public Shared Sub Main()

        ' Измените имя приложения в случае необходимости
        Dim RunningProcesses As Process() = _
             Process.GetProcessesByName("OneInstance")

        If (RunningProcesses.Length = 1) Then
            Application.Run(New Form1)
```

```
        Else
            ShowWindowAsync( _
                       RunningProcesses(0).MainWindowHandle, _
                       ShowWindowConstants.SW_SHOWMINIMIZED)
            ShowWindowAsync( _
                      RunningProcesses(0).MainWindowHandle, _
                      ShowWindowConstants.SW_RESTORE)
        End If
    End Sub
```

В этом примере мы получаем имя процесса с помощью класса `Process` и обрабатываем ситуацию. Если запускается первый экземпляр формы, то длина массива `RunningProcesses` равна единице, и мы запустим программу в обычном режиме. В противном случае мы просто активизируем запущенную ранее программу. Обратите внимание на интересное применение констант Windows API. Вместо обычного объявления констант в примере используется перечисление типа `Enum` — `ShowWindowConstants`. Если вы будете писать вышеприведенный код вручную, то заметите, что у вас будет работать всплывающая подсказка с перечислением имеющихся констант. Использование подобного приема порой облегчает написание кода сложных приложений.

19.5. Замена обоев на рабочем столе

Существует целый класс программ, которые по расписанию меняют картинки на рабочем столе. Подобные картинки называют *обоями* Windows. В Интернете есть целые сайты, на которых собраны тысячи красивых картинок, специально подготовленных для размещения на рабочем столе. Я покажу, как можно самому программно установить вашу картинку с изображением любимого кота в качестве обоев. Создадим новый проект с именем `Wallpaper`. Для него вполне достаточно одной кнопки. Так как в нашем случае кнопка не играет никакой роли, то оставим все ее свойства по умолчанию. Переходим в редактор кода. Вначале надо объявить функцию Windows API `SystemParametersInfo` и несколько констант, используемых функцией (листинг 19.7).

Листинг 19.7. Замена обоев на рабочем столе

```
'--------------------------------------
' Замена обоев на рабочем столе © 2004 Александр Климов
'--------------------------------------
```

```
   Private Const SPI_SETDESKWALLPAPER As Integer = &H14
   Private Const SPIF_UPDATEINIFILE As Integer = &H1
   Private Const SPIF_SENDWININICHANGE As Integer = &H2
   Private Declare Auto Function SystemParametersInfo Lib "user32.dll" _
       ( ByVal uAction As Integer, ByVal uParam As Integer, _
         ByVal lpvParam As String, ByVal fuWinIni As Integer) As Integer

   Dim imageLocation As String = Application.StartupPath &
"..\..\mycat.bmp"

     Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
        Try
           SystemParametersInfo(SPI_SETDESKWALLPAPER, 0, imageLocation, _
                SPIF_UPDATEINIFILE Or SPIF_SENDWININICHANGE)
        Catch Ex As Exception
            MsgBox("Ошибка при установке обоев: " & Ex.Message)
        End Try
    End Sub
```

В случае необходимости определите путь к вашему файлу. Обратите внимание, что в качестве обоев надо использовать картинки формата BMP. Также желательно подогнать заранее размер картинки под размер вашего монитора. В этом примере я организовал небольшую проверку на возможную ошибку. Сама установка картинки в качестве обоев заключается в вызове функции `SystemParametersInfo`. Параметры `SPIF_UPDATEINIFILE` и `SPIF_SENDWININICHANGE` позволяют оповестить Windows о смене обоев и дают ей команду на обновление картинки на рабочем столе. Как видите, ничего сложного здесь нет. Запустите программу на исполнение. Закройте все остальные окна программы, если они у вас запущены, оставив на экране только окно формы нашего проекта, за которым можно видеть рабочий стол. Теперь при нажатии на кнопку вы сразу увидите результат.

19.6. Анимированные курсоры

Вероятно, вам приходилось видеть в Windows то, что фирма Microsoft предпочитает называть *подвижными указателями*. На мой взгляд, не самое удачное название. Я буду использовать более понятное и традиционное — анимированные курсоры. К сожалению, класс `Cursor` не поддерживает ни анимированные, ни цветные курсоры. Но проблему легко решить путем вы-

зова функции Windows API `LoadCursorFromFile`. Для начала надо объявить ее соответствующим образом — проект `AniCursor` (листинг 19.8).

Листинг 19.8. Использование анимированных курсоров

```
'--------------------------------------
' Анимированные курсоры © 2004 Александр Климов
'--------------------------------------
Imports System.IO

Private Declare Unicode Function LoadCursorFromFile _
     Lib "user32.dll" _
     Alias "LoadCursorFromFileW" (ByVal filename As String) As IntPtr

Private Const animatcursor As String = "cat.ani"

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
        Dim filename As String
        Dim hCursor As IntPtr
        filename = Path.Combine(Application.StartupPath, animatcursor)
        hCursor = LoadCursorFromFile(filename)
        Me.Cursor = New Cursor(hCursor)
    End Sub

    Private Sub Form1_Click(ByVal sender As Object, ByVal e As
System.EventArgs) Handles MyBase.Click
        Me.Cursor = Cursors.Default
    End Sub
```

Запустите пример. При нажатии на кнопку курсор вашей мыши изменится на изображение бегущего котенка. Этот курсор будет работать в пределах клиентской области формы. Чтобы вернуть курсору прежний вид, достаточно установить его свойство по умолчанию. В нашем примере, для этого достаточно щелчка мыши на форме (событие `Form1_Click`).

19.7. Необычная каретка

В текстовых полях применяются особые виды курсоров, больше известные как *каретки*. Как правило, каретка отображается тонкой вертикальной чер-

точкой. Это свойство заложено по умолчанию для текстовых полей, но при желании можно изменить высоту и толщину каретки и даже использовать значок! В проекте `Caret` я продемонстрирую этот способ (листинг 19.9).

Листинг 19.9. Использование значка в каретке

```
'--------------------------------
' Каретка © 2005 Александр Климов
'--------------------------------
    Declare Function CreateCaret Lib "user32" _
                (ByVal hwnd As IntPtr, ByVal hBitmap As IntPtr, _
                 ByVal nWidth As Integer, ByVal nHeight As Integer) As
Integer
    Declare Function DestroyCaret Lib "user32.dll" () As Integer
    Declare Function SetCaretPos Lib "user32.dll" _
                    (ByVal x As Integer, ByVal y As Integer) As Integer
    Declare Function HideCaret Lib "user32.dll" _
                    (ByVal hwnd As IntPtr) As Integer
    Declare Function ShowCaret Lib "user32" _
          (ByVal hwnd As IntPtr) As Integer


    Dim bm As Bitmap = New Bitmap(Application.StartupPath &
"..\..\sound.ico")
    Dim hBitmap As System.IntPtr = bm.GetHbitmap()


    Private Sub TextBox1_GotFocus(ByVal sender As Object, ByVal e As
System.EventArgs) Handles TextBox1.GotFocus
        CreateCaret(TextBox1.Handle, hBitmap, 16, 16)
        ShowCaret(TextBox1.Handle)
    End Sub
```

Для начала надо поместить на форму текстовое поле, которое будет служить объектом исследования. В редакторе кода объявляем несколько функций Windows API, которые используются при работе с кареткой. Реально в моем примере используются только две функции, остальные я оставил про запас на тот случай, если вы захотите расширить возможности примера. Так как мы договорились использовать в качестве каретки значок, необходимо указать программе путь к нему. Из эстетических соображений я выбрал значок размером 16×16 пикселов, но это совсем не обязательно. Надо сказать, что Windows упрямо пытается восстановить прежний вид каретки, поэтому приходится постоянно вызывать пару функций. Самое удобное место для вызо-

ва — событие GotFocus. Должен предупредить, что текстовые поля не поддерживают полноценные цветные значки, они выводятся немного искаженными, поэтому, если уж вы решили использовать в своих программах значки, то позаботьтесь об их внешнем виде.

19.8. Разноцветная консоль

В некоторых случаях гораздо удобнее использовать консольные приложения вместо форм с графическим интерфейсом. Но стандартный вид консоли с черным экраном и белыми буквами наводит скуку. А что если использовать другие цвета? Никогда об этом не задумывались? А зря! Хочу рассказать, как разукрасить консоль разными цветами. На данный момент .NET Framework не позволяет напрямую менять цвета консоли. И здесь нам на помощь опять приходят функции Windows API. Запускаем новый проект и выбираем тип консольного приложения. Добавим новый класс к созданному проекту. По умолчанию его имя Class1.vb. В этом классе мы поместим вызовы нужных нам функций — проект ColorConsole (листинг 19.10).

Листинг 19.10. Создание разноцветной консоли

```
'-----------------------------------------
' Разноцветная консоль © 2004 Александр Климов
'-----------------------------------------
Imports System.Runtime.InteropServices

 Public Class Class1

    Private hConsoleHandle As IntPtr
    Private ConsoleOutputLocation As COORD
    Private ConsoleInfo As CONSOLE_SCREEN_BUFFER_INFO
    Private OriginalColors As Integer

    Private Const STD_OUTPUT_HANDLE As Integer = &HFFFFFFF5

    Private Declare Auto Function GetStdHandle Lib "kernel32.dll"  _
        ByVal nStdHandle As Integer) As IntPtr

    Private Declare Auto Function GetConsoleScreenBufferInfo _
         Lib "kernel32.dll" (ByVal hConsoleOutput As IntPtr, _
                             ByRef lpConsoleScreenBufferInfo As _
                             CONSOLE_SCREEN_BUFFER_INFO) As Integer
```

```
Private Declare Auto Function SetConsoleTextAttribut _
     Lib "kernel32" ByVal hConsoleOutput As IntPtr _
     ByVal wAttributes As Integer) As Long

Public Enum Foreground
   Blue = &H1
   Green = &H2
   Red = &H4
   Intensity = &H8
End Enum

Public Enum Background
   Blue = &H10
   Green = &H20
   Red = &H40
   Intensity = &H80
End Enum

<StructLayout(LayoutKind.Sequential)> _
Private Structure COORD
   Dim X As Short
   Dim Y As Short
End Structure

<StructLayout(LayoutKind.Sequential)> _
Private Structure SMALL_RECT
   Dim Left As Short
   Dim Top As Short
   Dim Right As Short
   Dim Bottom As Short
End Structure

<StructLayout(LayoutKind.Sequential)> _
Private Structure CONSOLE_SCREEN_BUFFER_INFO
   Dim dwSize As COORD
   Dim dwCursorPosition As COORD
   Dim wAttributes As Integer
   Dim srWindow As SMALL_RECT
```

```
        Dim dwMaximumWindowSize As COORD
    End Structure

    Sub New()
        hConsoleHandle = GetStdHandle(STD_OUTPUT_HANDLE)
        GetConsoleScreenBufferInfo(hConsoleHandle, ConsoleInfo)
        OriginalColors = ConsoleInfo.wAttributes
    End Sub

    Public Sub TextColor(ByVal Colors As Integer)
        ' Устанавливаем цвет для символов
        SetConsoleTextAttribute(hConsoleHandle, Colors)
    End Sub

    Public Sub ResetColor()
        ' Восстанавливаем оригинальные цвета
        SetConsoleTextAttribute(hConsoleHandle, OriginalColors)
    End Sub
End Class
```

Теперь переходим в основной модуль `Module1.vb` и пишем код там (листинг 19.11).

Листинг 19.11. Использование цветной консоли

```
'-----------------------------------------
' Разноцветная консоль © 2004 Александр Климов
'-----------------------------------------
Imports System.Console
Module Module1

    Sub Main()
        Dim TextChange As New Class1
        WriteLine("Оригинальный цвет")
        WriteLine("Нажмите ENTER для начала демонстрации примера")
        ReadLine()

        TextChange.TextColor(_
          Class1.Foreground.Green + Class1.Foreground.Intensity)
```

```
        WriteLine("Текст выводится зеленым цветом")
        WriteLine("Нажмите ENTER для изменения цвета текста")
        ReadLine()
        TextChange.TextColor(_
          Class1.Foreground.Red + Class1.Foreground.Intensity)
        WriteLine("Теперь текст  красный")
        WriteLine("Нажмите ENTER для нового изменения цвета")
        ReadLine()
        TextChange.TextColor(_
          Class1.Foreground.Blue + _
          Class1.Foreground.Intensity + _
          Class1.Background.Green + _
          Class1.Background.Intensity)
        WriteLine("Теперь текст синий, а фон под текстом зеленый")
        WriteLine("Нажмите ENTER для восстановления прежних цветов")
        ReadLine()
        TextChange.ResetColor()
        WriteLine("Настройки цветов восстановлены")
        WriteLine("Нажмите ENTER для выхода")
        ReadLine()

    End Sub
End Module
```

Пример очень простой и дополнительных объяснений не требует.

Примечание

В версии Visual Basic .NET 2005 разработчики обещали добавить новые классы для работы с консольными приложениями, где будут реализованы возможности, описанные в примере.

19.9. Получение информации о версии программы

Как правило, в диалоговом окне **О программе** автор указывает свое имя, название компании, торговую марку, авторское право и версию программы. Можно вручную прописывать эти данные в коде программы. Но, как правило, так никто не делает. Существует возможность автоматизации этой рутинной работы. Когда вы создаете новый проект в Visual Basic

.NET 2003, то наряду с другими файлами на диске создается специальный файл AssemblyInfo.vb, предназначенный для хранения данных такого типа. Откройте этот файл в редакторе кода, и вы увидите приблизительно такой текст:

```
Imports System
Imports System.Reflection
Imports System.Runtime.InteropServices


' General Information about an assembly is controlled through the
following
' set of attributes. Change these attribute values to modify the
information
' associated with an assembly.


' Review the values of the assembly attributes


<Assembly: AssemblyTitle("")>
<Assembly: AssemblyDescription("")>
<Assembly: AssemblyCompany("")>
<Assembly: AssemblyProduct("")>
<Assembly: AssemblyCopyright("")>
<Assembly: AssemblyTrademark("")>
<Assembly: CLSCompliant(True)>


'The following GUID is for the ID of the typelib if this project is
exposed to COM
<Assembly: Guid("AD5CF26B-20A8-44E7-83EB-C08C8BB74528")>


' Version information for an assembly consists of the following four
values:
'
'      Major Version
'      Minor Version
'      Build Number
'      Revision
'
' You can specify all the values or you can default the Build and
Revision Numbers
' by using the '*' as shown below:


<Assembly: AssemblyVersion("1.0.*")>
```

Вы можете модифицировать некоторые значения. Если оставить без изменения ряд параметров, то будут использоваться значения по умолчанию. Вот как изменил я некоторые строчки для нашего примера:

```
<Assembly: AssemblyTitle("Информация о сборке программы")>
<Assembly: AssemblyDescription("Пример получения информации о версии
программы")>
<Assembly: AssemblyCompany("Александр Климов")>
<Assembly: AssemblyVersion("2.0.*")>
```

Установив новые значения для сборки проекта, можно получать эти данные из программы. Расположите на форме несколько надписей, в которых будут отображаться требуемые значения. Вот полный код получения различной информации из программы — проект `Version` (листинг 19.12).

Листинг 19.12. Получение информации о версии программы

```
'--------------------------------------
' Получение информации о версии программы © 2004 Александр Климов
'--------------------------------------
Imports System.Reflection


    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        MyAssembly = [Assembly].GetExecutingAssembly()


        lblGetDescr.Text = GetAssemblyDescription()
        lblGetVersion.Text = GetAssemblyVersion()
        lblGetCompany.Text = GetAssemblyCompany()
    End Sub


    ' Возвращает значение атрибута Description.
    Private Function GetAssemblyDescription() As String
        If MyAssembly.IsDefined(GetType(AssemblyDescriptionAttribute),
        True) Then
            Dim attr As Attribute = _
                Attribute.GetCustomAttribute(MyAssembly, _
                    GetType(AssemblyDescriptionAttribute))
            Dim description_attr As AssemblyDescriptionAttribute = _
                DirectCast(attr, AssemblyDescriptionAttribute)
            Return description_attr.Description
        Else
```

```
            Return ""
        End If
    End Function

    ' Возвращает значение атрибута Company.
    Private Function GetAssemblyCompany() As String
        If MyAssembly.IsDefined(GetType(AssemblyCompanyAttribute), True)
Then
            Dim attr As Attribute = _
                Attribute.GetCustomAttribute(MyAssembly, _
                    GetType(AssemblyCompanyAttribute))
            Dim company_attr As AssemblyCompanyAttribute = _
                DirectCast(attr, AssemblyCompanyAttribute)
            Return company_attr.Company
        Else
            Return ""
        End If
    End Function

    ' Возвращает значение атрибута Version.
    Private Function GetAssemblyVersion() As String
        Dim myVersion As AssemblyName = MyAssembly.GetName()

        Return myVersion.Version.Major.ToString & "." & _
            myVersion.Version.Minor.ToString & "." & _
            myVersion.Version.Build.ToString & "." & _
            myVersion.Version.Revision.ToString
    End Function
```

19.10. Конвертер римских чисел

Мы часто используем в своей повседневной жизни римские числа — на циферблатах часов, на страницах книг, на памятниках. В Москве есть станция метро "Римская", построенная при участии итальянских строителей. Над вестибюлем можно увидеть дату открытия станции — MCMXCV. Интересно, вы сможете с ходу определить записанный год? У некоторых людей вызывает затруднение запись чисел в римской системе счисления. Действительно, она не очень удобна и громоздка. Но образованному человеку необходимо иметь о ней хотя бы минимум знаний. В римской системе существует следующее правило: "Если бóльшая цифра находится перед мень-

шей, то они складываются. Если меньшая цифра стоит перед большей, то она вычитается из большей". Вот несколько примеров:

```
VI = 5 + 1 = 6
IXX = 20 - 1 = 19
```

Американский ученый Стивен Шварцман предложил специальный стандарт для римских чисел (ISRN, International Standart Roman Numerals), в котором описаны несколько правил их записи. Например, нельзя записывать одну и ту же цифру более трех раз подряд. Поэтому не удивляйтесь, если увидите на старинных часах числа, не соответствующие стандарту, который был предложен относительно недавно. В частности, нередко встречается запись числа четыре как IIII вместо привычного уже нам IV. В табл. 19.1 представлено соответствие римских чисел применяемым нами арабским.

Таблица 19.1. *Соответствие римских чисел арабским*

Единицы		Десятки		Сотни		Тысячи	
1	I	10	X	100	C	1000	M
2	II	20	XX	200	CC	2000	MM
3	III	30	XXX	300	CCC	3000	MMM
4	IV	40	XL	400	CD		
5	V	50	L	500	D		
6	VI	60	LX	600	DC		
7	VII	70	LXX	700	DCC		
8	VIII	80	LXXX	800	DCCC		
9	IX	90	XC	900	CM		

Но держать в голове подобную таблицу нет необходимости. Мы с вами напишем конвертер римских чисел в стандартный вид (проект `Roman2Arab`). Для нашего проекта понадобятся два текстовых поля и одна кнопка. Для первого текстового поля, в которое будут вводиться римские числа, установим следующие свойства:

- для первого текстового поля:
  - `Name = txtRoman`;
  - `CharacterCasing = Upper`;
  - `Text = X`;

- для второго текстового поля, в котором будут выводиться арабские цифры:
  - `Name` = `txtArab`;
  - `ReadOnly` = `True`;
  - `Text` = `10`;
- для кнопки достаточно установить два свойства:
  - `Name` = `butConvert`;
  - `Text` = `Конвертировать`.

Небольшие пояснения к установленным свойствам. Так как римские цифры принято записывать заглавными буквами латинского алфавита, то я установил свойство `CharacterCasing` в `Upper`, что позволяет автоматически переводить символы в верхний регистр. Теперь запишем небольшую функцию для перевода римского числа в обычный вид (листинг 19.13).

Листинг 19.13. Конвертер римских чисел в стандартный вид

```
'--------------------------------------
' Конвертер римских чисел © 2004 Александр Климов
'--------------------------------------
    ' Возвращает число в арабских цифрах
    Private Function RomanToArabic(ByVal roman As String) As Long
        Dim i As Integer
        Dim ch As String
        Dim result As Long
        Dim new_value As Long
        Dim old_value As Long

        old_value = 1000

        For i = 1 To Len(roman)
            ch = Mid$(roman, i, 1)
            Select Case ch
                Case "I"
                    new_value = 1
                Case "V"
                    new_value = 5
                Case "X"
                    new_value = 10
```

```
                Case "L"
                    new_value = 50
                Case "C"
                    new_value = 100
                Case "D"
                    new_value = 500
                Case "M"
                    new_value = 1000
            End Select

            ' Если символ больше предыдущего
            If new_value > old_value Then
                ' Если new value > предыдущего символа,
                ' то добавляем это значение к результату
                ' и вычитаем дважды предыдущий символ.
                result = result + new_value - 2 * old_value
            Else
                ' Если new value <= предыдущего символа,
                ' то добавляем его к результату.
                result = result + new_value
            End If

            old_value = new_value
        Next i
        Return result
    End Function

    Private Sub butConvert_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles butConvert.Click
        txtArab.Text = RomanToArabic(txtRoman.Text)
    End Sub
```

Если у вас возникли затруднения с вычислением даты открытия станции "Римская", то теперь вы можете проверить свои вычисления с помощью написанного конвертера.

19.11. Свойства файла

Если в Проводнике щелкнуть правой кнопкой мыши на файле и выбрать из контекстного меню пункт **Свойства**, то перед вами предстанет диалоговое

окно с перечислением различных свойств файла. Можно самому реализовать подобное окно для получения свойств файла. Все необходимые свойства файла содержатся в классе `FileVersionInfo` (проект `FileProperties`). Разместим на форме группу надписей и текстовых полей, в которых будут содержаться описания получаемых свойств и сами свойства. В верхней части формы поместим еще одно текстовое поле, в котором вы сможете впечатать свой путь к файлу. Для получения свойств файла необходимо нажать на кнопку (листинг 19.14).

Листинг 19.14. Получение свойств файла

```
'---------------------------------------
' Получение свойств файла © 2004 Александр Климов
'---------------------------------------
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
        ' Получим свойства файла из заданного файла
        Dim FileProperties As FileVersionInfo = FileVersion-
Info.GetVersionInfo(txtFilePath.Text)

        ' Имя файла
        TextBox1.Text = FileProperties.FileDescription
        ' Версия файла
        TextBox2.Text = FileProperties.FileVersion
        ' Внутреннее имя файла
        TextBox3.Text = FileProperties.InternalName
        ' Исходное имя файла
        TextBox4.Text = FileProperties.OriginalFilename
        ' Имя продукта
        TextBox5.Text = FileProperties.ProductName
        ' Версия продукта
        TextBox6.Text = FileProperties.ProductVersion
        ' Язык
        TextBox7.Text = FileProperties.Language
    End Sub
```

Теперь вы сами можете программным путем узнавать необходимые свойства файла.

19.12. Скриншоты

Мы уже рассматривали в *главе 9* тему получения снимка экрана или активного окна с помощью программного нажатия на клавишу <Print Screen> (проект `PrintScreen`). Но данный способ не обладает достаточной гибкостью. Поэтому вернемся к этой теме снова и изучим другие способы получения скриншотов. Напомню, что *скриншотами* называют снимки, сделанные с экрана монитора. Наиболее широко они используются в игровых журналах. По этим снимкам геймер может получить приблизительное представление о новой игре. Также широко скриншоты используются на сайтах для ознакомления с внешним видом новой версии программы. Существует несколько типичных вариантов скриншотов: снимок всего экрана, снимок формы, снимок отдельного элемента управления. Я хочу дать вам представление о том, как самому программно делать подобные снимки. Пример реализован в проекте `GetScreenShot`. Нам понадобятся одно графическое поле и три кнопки. Я не стал присваивать им говорящие имена, только изменил тексты для кнопок, чтоб не запутаться. Сначала приведу полный листинг программы (листинг 19.15).

Листинг 19.15. Получение скриншотов

```
'---------------------------------------
' Получение скриншотов © 2004 Александр Климов
'---------------------------------------
    ' Структуры Windows API
    Private Structure POINTAPI
        Public x As Int32
        Public y As Int32
    End Structure

    Private Structure RECT
        Public Left As Int32
        Public Top As Int32
        Public Right As Int32
        Public Bottom As Int32
    End Structure

    ' Функции API
    Private Declare Function GetDesktopWindow Lib "user32.dll" _
            () As IntPtr
```

```
    Private Declare Function GetWindowRect Lib "user32.dll" _
           (ByVal hWnd As IntPtr, ByRef lpRect As RECT) As Int32

    Private Declare Function ScreenToClient Lib "user32.dll" _
           (ByVal hWnd As IntPtr, ByRef lpPoint As POINTAPI) As Int32

    Private Declare Function GetDC Lib "user32.dll" _
             (ByVal hWnd As IntPtr) As IntPtr

    Private Declare Function BitBlt Lib "gdi32" Alias "BitBlt" _
            (ByVal hDestDC As Integer, ByVal x As Integer, _
             ByVal y As Integer, ByVal nWidth As Integer, _
             ByVal nHeight As Integer, ByVal hSrcDC As Integer, _
             ByVal xSrc As Integer, ByVal ySrc As Integer, _
             ByVal dwRop As Integer) As Integer

  Private Declare Function GetWindowDC Lib "user32" Alias "GetWindowDC" _
           (ByVal hwnd As Integer) As IntPtr

    Private Declare Function ReleaseDC Lib "user32.dll" _
           (ByVal hWnd As IntPtr, ByVal hdc As IntPtr) As Int32

    Private Declare Function StretchBlt Lib "gdi32.dll" _
           (ByVal hdc As IntPtr, ByVal x As Int32, _
            ByVal y As Int32, ByVal nWidth As Int32, _
            ByVal nHeight As Int32, ByVal hSrcDC As IntPtr, _
            ByVal xSrc As Int32, ByVal ySrc As Int32, _
            ByVal nSrcWidth As Int32, ByVal nSrcHeight As Int32, _
            ByVal dwRop As Int32) As Int32

    ' Константа
    Private Const SRCCOPY As Int32 = &HCC0020

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
        ' Получим снимок всего экрана
        Dim hWndDesktop As IntPtr

        ' Получим дескриптор рабочего стола
        hWndDesktop = GetDesktopWindow()
```

```
    ' Получим размеры окна
    Dim rct As RECT
    GetWindowRect(hWndDesktop, rct)
    Dim Width As Int32 = rct.Right - rct.Left
    Dim Height As Int32 = rct.Bottom - rct.Top

    ' Координаты в клиентской системе координат
    Dim pt As New POINTAPI
    pt.y = rct.Top
    pt.x = rct.Left
    ScreenToClient(hWndDesktop, pt)
    rct.Left = pt.x
    rct.Top = pt.y

    ' Объект Bitmap, в котором будет содержаться скриншот экрана
    Dim b As New Bitmap(Width, Height)

    ' Создадим объект Graphics
    Dim g As Graphics = Graphics.FromImage(b)

    ' Получим контекст устройства экрана
    Dim hdcWindow As IntPtr = GetDC(hWndDesktop)

    ' Получим контекст устройства объекта Graphics
    Dim hdc As IntPtr = g.GetHdc()

    BitBlt(hdc.ToInt32, 0, 0, _
       Width, Height, hdcWindow.ToInt32, _
       rct.Left, rct.Top, SRCCOPY)

    ' Освобождаем ресурсы
    ReleaseDC(hWndDesktop, hdcWindow)
    g.ReleaseHdc(hdc)

    ' Помещаем снимок в графический объект
    PictureBox1.Image = b
End Sub
```

```
    Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button2.Click
        ' Скриншот кнопки
        ' Получим контекст устройства кнопки через его дескриптор
        Dim hdcWindow As IntPtr = GetDC(Button1.Handle)
        Dim rct As RECT
        Dim pt As New POINTAPI
        pt.y = rct.Top
        pt.x = rct.Left
        ScreenToClient(hdcWindow, pt)
        rct.Left = pt.x
        rct.Top = pt.y

        Dim b As New Bitmap(PictureBox1.Width, PictureBox1.Height)

        Dim g As Graphics = Graphics.FromImage(b)
        Dim hdc As IntPtr = g.GetHdc()
        BitBlt(hdc.ToInt32, 0, 0 _
           Button1.Width, Button1.Height, hdcWindow.ToInt32, _
           rct.Left, rct.Top, SRCCOPY)

        PictureBox1.Image = b
        ReleaseDC(Button1.Handle, hdcWindow)
        g.ReleaseHdc(hdc)
        g.Dispose()

    End Sub

    Private Sub Button3_Click(ByVal sender As System.Object, _
          ByVal e As System.EventArgs) Handles Button3.Click

        Dim hdcForm As IntPtr = GetWindowDC(Me.Handle.ToInt32)
        Dim rct As RECT
        Dim pt As New POINTAPI
        pt.y = rct.Top
        pt.x = rct.Left
        ScreenToClient(hdcForm, pt)
        rct.Left = pt.x
        rct.Top = pt.y
```

```
        Dim b As New Bitmap(PictureBox1.Width, PictureBox1.Height)

        Dim g As Graphics = Graphics.FromImage(b)
        Dim hdc As IntPtr = g.GetHdc()

        StretchBlt(hdc, 0, 0, _
           PictureBox1.Width, PictureBox1.Height, _
           hdcForm, rct.Left, rct.Top, Width, Height, SRCCOPY)

        PictureBox1.Image = b
        ReleaseDC(Button1.Handle, hdcForm)
        g.ReleaseHdc(hdc)
        g.Dispose()

    End Sub

    Private Sub Button4_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button4.Click
        ' Делаем снимок экрана через определенное время

        ' Меняем указатель мыши
        Me.Cursor = Cursors.WaitCursor

        ' Задаем время 3 секунды
        Timer1.Interval = 3000
        Timer1.Enabled = True
    End Sub

    Private Sub Timer1_Tick(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Timer1.Tick
        Timer1.Enabled = False
        Dim hWndDesktop As IntPtr

        hWndDesktop = GetDesktopWindow()
        ' Получим контекст устройства экрана
        Dim hdcWindow As IntPtr = GetDC(hWndDesktop)

        Dim desk_bounds As Rectangle
        desk_bounds = Screen.GetBounds(New Point(1, 1))
```

```
        Dim desk_w As Int32 = desk_bounds.Width
        Dim desk_h As Int32 = desk_bounds.Height

        ' Объект Bitmap, в котором будет содержаться скриншот экрана
        Dim b As New Bitmap(desk_w, desk_h)

        ' Создадим объект Graphics
        Dim g As Graphics = Graphics.FromImage(b)

        ' Получим контекст устройства объекта Graphics
        Dim hdc As IntPtr = g.GetHdc()

        ' Получим изображение
        StretchBlt( _
            hdc, 0, 0, desk_w, desk_h, _
            hdcWindow, 0, 0, desk_w, desk_h, _
            SRCCOPY)

        ' Освобождаем ресурсы
        ReleaseDC(hWndDesktop, hdcWindow)
        g.ReleaseHdc(hdc)

        ' Помещаем снимок в графический объект
        PictureBox1.Image = b
        ' Восстанавливаем курсор мыши на стандартный указатель
        Me.Cursor = Cursors.Default
    End Sub
```

Перейдем к разбору примера. Сначала нужно объявить несколько структур и функций Windows API, так как стандартными средствами мы не сможем добиться нужного результата. Теперь можно приступать к реализации задуманного. В первом случае нам предстоит сделать снимок всего экрана. Для этого необходимо получить дескриптор экрана через вызов функции `GetDesktopWindow`. Имея данный дескриптор, можно получить размеры экрана через функцию `GetWindowRect`. При этом следует произвести ряд несложных математических вычислений:

```
' Получим размеры окна
Dim Width As Int32 = rct.Right - rct.Left
Dim Height As Int32 = rct.Bottom - rct.Top
```

Получив необходимые размеры, переводим их в клиентскую систему координат через вызов функции `ScreenToClient`. Далее создаем объекты `Bitmap` и `Graphics`, которые позволят нам работать с графикой. Имея дескриптор окна и графические объекты, можно создавать контекст устройства, который непосредственно работает с графикой при помощи функции `BitBlt`. Эта функция позволяет копировать прямоугольную область из контекста одного устройства в контекст другого. Полученный снимок мы помещаем в графическое поле `PictureBox`. Не забывайте освобождать ресурсы компьютера после проделанной работы:

```
ReleaseDC(hWndDesktop, hdcWindow)
g.ReleaseHdc(hdc)
```

19.12.1. Скриншот отдельного элемента

Используя аналогичную технику, можно получить снимок не только всего экрана, но и отдельного элемента. Например, вторая кнопка на форме позволяет получить изображение первой кнопки. В этом случае у нас нет необходимости прибегать к помощи функции Windows API для получения дескриптора, так как у кнопки уже имеется нужное свойство `Button1.Handle`. Далее код практически не отличается от получения снимка экрана. Запустив проект, нажмите на вторую кнопку, чтобы получить скриншот отдельной кнопки.

19.12.2. Скриншот формы

При получении скриншота формы есть одна тонкость. Дело в том, что если воспользоваться вызовом функции `GetDC`, то мы получим изображение только клиентской области формы. Чтобы обойти это ограничение и получить всю форму, включая заголовок, меню и полосы прокрутки, необходимо использовать функцию `GetWindowDC`. Далее все то же самое. Для разнообразия я использовал схожую функцию `StrectcBlt`, позволяющую растягивать или сжимать изображения.

19.12.3. Снимок через заданный интервал

Четвертая кнопка позволяет делать снимки через определенный промежуток времени. Такой снимок может понадобиться, если вы хотите снять, например, окно программы Менеджер задач, которое открывается при нажатии комбинации клавиш <Alt>+<Tab>. Когда вы отпустите клавиши, окно сразу же исчезнет, и вы не сможете получить его снимок. Решить эту проблему можно с помощью таймера. Устанавливаем интервал в три секунды, нажимаем нужную комбинацию клавиш. Через отведенный промежуток времени

снимок экрана с окном Менеджера задач появится в графическом объекте PictureBox.

19.13. Семь пятниц на неделе

Вы знаете, как по-арабски пишется слово "Пятница"? А по-немецки? Ну, хотя бы по-украински вы сможете написать? Совсем не обязательно лезть в иностранные словари, чтобы узнать начертание этого слова. В составе VB .NET присутствует класс CultureInfo, с помощью которого мы и узнаем, как пишется слово "Пятница" на семи разных языках. Для проекта Friday нам понадобятся семь надписей Label, семь текстовых полей TextBox и одна кнопка (листинг 19.16).

Листинг 19.16. Перевод слова "Пятница" на разные языки

```
'---------------------------------------
' Семь пятниц © 2004 Александр Климов
'---------------------------------------
Imports System.Globalization

    Private Sub butFriday_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles butFriday.Click
        ' По-русски
        Dim ci As New CultureInfo("ru-RU")
        ' Получим объект DateTimeFormatInfo
        Dim dtfi As DateTimeFormatInfo = ci.DateTimeFormat
        TextBox1.Text = (dtfi.GetDayName(DayOfWeek.Friday))

        'По-египетски
        ci = New CultureInfo("ar-EG")
        dtfi = ci.DateTimeFormat
        TextBox2.Text = (dtfi.GetDayName(DayOfWeek.Friday))

        'По-белорусски
        ci = New CultureInfo("be-BY")
        dtfi = ci.DateTimeFormat
        TextBox3.Text = (dtfi.GetDayName(DayOfWeek.Friday))

        'По-болгарски
        ci = New CultureInfo("bg-BG")
```

```
        dtfi = ci.DateTimeFormat
        TextBox4.Text = (dtfi.GetDayName(DayOfWeek.Friday))

        'По-фински
        ci = New CultureInfo("fi-FI")
        dtfi = ci.DateTimeFormat
        TextBox5.Text = (dtfi.GetDayName(DayOfWeek.Friday))

        'По-французски
        ci = New CultureInfo("fr-FR")
        dtfi = ci.DateTimeFormat
        TextBox6.Text = (dtfi.GetDayName(DayOfWeek.Friday))

        'По-украински
        ci = New CultureInfo("uk-UA")
        dtfi = ci.DateTimeFormat
        TextBox7.Text = (dtfi.GetDayName(DayOfWeek.Friday))
    End Sub
```

С помощью класса `CultureInfo` мы задаем нужный язык, а затем переводим константу `DayOfWeek.Friday`, соответствующую пятнице, на заданный язык (рис. 19.1). Таким же способом можно перевести и многие другие константы, определяющие названия месяцев и дней недели.

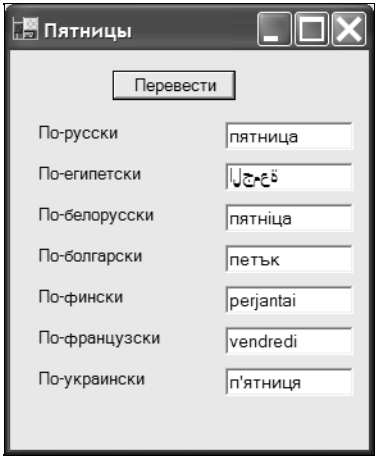

Рис. 19.1. Перевод слова на разные языки

19.14. Определение версии Windows

С выходом новой версии Windows у программистов добавляется хлопот по обеспечению совместимость своих программ. Появляются новые функции и, наоборот, исчезают старые. Чтобы разработанная вами программа корректно работала под всеми версиями Windows, приходится учитывать текущую версию Windows у пользователя. Для решения такой задачи существует класс `OperatingSystem` пространства имен `System`. Свойства этого класса содержат всю необходимую информацию для определения платформы компьютера. Можно воспользоваться следующими данными (табл. 19.2).

Таблица 19.2. *Свойства класса* `OperatingSystem` *для определения платформы компьютера*

OS	Платформа	Major	Minor
Win95	1	4	0
Win98	1	4	10
WinME	1	4	90
NT 3.51	2	3	51
NT 4.0	2	4	0
Win2000	2	5	0

Для определения версии Windows можно воспользоваться следующим примером — проект `WinVersion` (листинг 19.17).

Листинг 19.17. Определение версии Windows

```
'--------------------------------------
' Определение версии Windows © 2005 Александр Климов
'--------------------------------------
Imports System.Environment

    Private osInfo As OperatingSystem

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
        MsgBox(GetWinVersion)
    End Sub
```

```
Public Function GetWinVersion() As String
    'Функция определения версии Windows
    osInfo = OSVersion
    With osInfo
        Select Case .Platform

            Case .Platform.Win32Windows
                Select Case (.Version.Minor)
                    Case 0
                        Return "Windows 95"
                    Case 10
                        If .Version.Revision.ToString() = "2222A"
                        Then
                            Return "Windows 98 Second Edition"
                        Else
                            Return "Windows 98"
                        End If
                    Case 90
                        Return "Windows Me"
                End Select

            Case .Platform.Win32NT
                Select Case (.Version.Major)
                    Case 3
                        Return "Windows NT 3.51"
                    Case 4
                        Return "Windows NT 4.0"
                    Case 5
                        If .Version.Minor = 0 Then
                            Return "Windows 2000"
                        Else
                            Return "Windows XP " &
                            .Version.Major.ToString & _
                                            "." &
                            .Version.Minor.ToString & _
                                            "." & .Version.Build & _
                                            "." & .Version.Revision

                        End If
                End Select
```

```
            Case Else
                GetWinVersion = "Ошибка"
        End Select
    End With
End Function
```

19.15. Письма

Если вам необходимо отправить письмо из вашего приложения, то можно воспользоваться пространством имен `System.Web.Mail`. Запустите новый проект под названием `WebSendMail`. Пространство имен `System.Web.Mail` по умолчанию не подключено к текущим проектам. Необходимо добавить на него ссылку вручную. Делается это следующим образом. Выберите в меню **Project** команду **Add Reference**. Откроется диалоговое окно **Add Reference**, в котором необходимо перейти на вкладку **.NET**. Найдите в списке элемент **System.Web.dll**, выделите его мышью и щелкните на кнопке **Select**. Щелкните на кнопке **OK**, чтобы закрыть диалоговое окно. На форму поместите кнопку, которая будет отвечать за отправку письма (листинг 19.18).

Листинг 19.18. Отправка письма из программы

```
'---------------------------------------
' Отправка писем © 2004 Александр Климов
'---------------------------------------
Imports System.Web.Mail

    Private Sub butSendMail_Click(ByVal sender As System.Object, _
                ByVal e As System.EventArgs) Handles butSendMail.Click
        Dim oMsg As MailMessage = New MailMessage

        ' Здесь должен быть ваш адрес
        oMsg.From = "cat2@mail.ru"

        ' Адрес человека, которому вы хотите послать письмо
        oMsg.To = "rusproject@mail.ru"

        ' Тема письма
        oMsg.Subject = "Одинокий кот желает познакомиться"
```

```
    ' Текст письма
    oMsg.Body = "Одинокий кот хочет познакомиться с кошкой" & _
                "О себе: рыжий, усатый, полосатый"

    ' Здесь должен быть ваш почтовый SMTP-сервер
    SmtpMail.SmtpServer = "smtp.mail.ru"
    SmtpMail.Send(oMsg)

    oMsg = Nothing
End Sub
```

Когда будете работать с примером, то измените строчку

```
oMsg.From = "cat2@mail.ru"
```

на ваш реальный электронный адрес. В качестве получателя я указал свой реальный почтовый адрес, в случае необходимости измените и его тоже. Также следует указать реальный почтовый сервер. В примере использован сервер популярной российской почтовой службы Mail.ru.

19.16. Сезам, откройся

Сейчас мы с вами напишем хакерскую программу. Цель ее — получить доступ к содержимому текста, скрытого за звездочками. Для программы нам понадобятся таймер `Timer1`, надпись `Label1` и текстовое поле `TextBox1` (проект `OpenPass`). Присвойте свойству `PasswordChar` текстового поля значение `PasswordChar = *`, а также включите таймер (`Enabled = True`). Остальные свойства можете оставить по умолчанию. Для получения текста, скрытого за звездочками, необходимо получить дескриптор окна, содержащего текст. В данном случае — дескриптор текстового поля. Имея в своем распоряжении необходимый дескриптор, можно получить текст окна при помощи функции `GetWindowText`. Так как мы с вами будем писать универсальную программу, способную получать дескрипторы чужих окон, то необходимо задействовать еще несколько функций Windows API. Функция `WindowsFromPoint` получает дескриптор, находящийся под указателем мыши, а функция `GetCursorPos` возвращает текущие координаты мыши в экранных координатах. Таким образом, наша задача сводится к следующему. Отслеживаем позицию мыши и получаем дескриптор окна под ее курсором. Далее просто извлекаем текст окна и помещаем его на всеобщее обозрение (листинг 19.19).

Листинг 19.19. Извлечение пароля

```
'--------------------------------------
' Получение текста за звездочками © 2004 Александр Климов
'--------------------------------------
    Structure POINTAPI
        Public x As Int32
        Public y As Int32
    End Structure

    Declare Function WindowFromPoint Lib "user32" _
       (ByVal xPoint As Integer, _
        ByVal yPoint As Integer) As Integer

    Declare Function GetCursorPos Lib "user32" _
           (ByRef lpPoint As POINTAPI) As Integer

    Public Declare Auto Function GetWindowText Lib "user32" _
           (ByVal hWnd As Integer, _
            ByVal lpString As System.Text.StringBuilder, _
            ByVal cch As Integer) As Integer

    Dim pt As POINTAPI

    Private Sub Timer1_Tick(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Timer1.Tick
        ' Получим текущую позицию курсора
        GetCursorPos(pt)

        ' Дескриптор окна под курсором
        Dim iret As Integer
        iret = WindowFromPoint(pt.x, pt.y)

        ' Буфер для строки
        Dim Caption As New System.Text.StringBuilder(256)

        ' Возвращаемый текст
        Dim t As String
        t = GetWindowText(iret, Caption, Caption.Capacity)
```

```
        ' Теперь буфер содержит текст окна
        Label1.Text = Caption.ToString
    End Sub
```

Запустите проект и подведите указатель мыши к тексту с звездочками. В надписи `Label1` отобразится данный текст в открытом виде (рис. 19.2)! Кстати, этим приемом я часто пользовался в Windows 95/98/ME и мог без труда вытащить забытые пароли к своей электронной почте или для доступа в Интернет. Впоследствии разработчики Microsoft переработали механизм скрытия секретных данных, и описанный способ уже не пройдет в Windows XP. Впрочем, если у вас еще сохранилась старая система, использующая текст со звездочками, то программа без труда получит скрытый текст. Но программу совсем не обязательно использовать как инструмент взлома забытых паролей. Просто поводите мышью над различными объектами рабочего стола. Вы увидите много интересной информации. Например, подведите мышь к кнопке **Пуск**, к панели быстрого запуска, к часикам в правом нижнем углу. Возможности программы расширяются путем добавления дополнительной функциональности. Например, можно извлекать не только текст окна, но и получить другую информацию (класс окна, его размеры и т. д.).

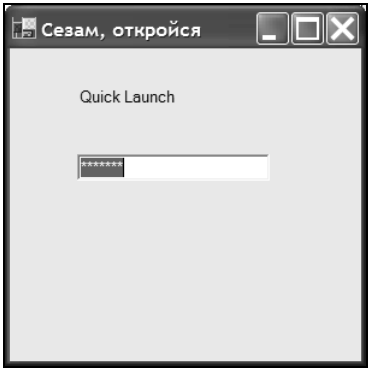

Рис. 19.2. Извлечение текста из других программ

Анекдот в тему

Программист говорит другому: "Мне сказали, что использовать имя моей кошки в качестве пароля — плохой тон. А я все равно не буду менять пароль! Я так привык к имени qZw813drgg1!"

19.17. Извлечение значков из файлов

Этот пример относится к числу "продвинутых" и позволяет извлекать значки из ресурсов, зашитых в файлы EXE, DLL, CPL и др. На данный момент GDI+ не позволяет извлекать значки с помощью управляемого кода, поэтому снова призовем к себе в помощники функции Windows API. Пример реализован в проекте `ExtractIcons`. Вначале объявим функцию `ExtractIcon`. В качестве испытуемого файла я выбрал программу Paint, входящую в состав Windows. Этот графический редактор содержится в файле mspaint.exe. В случае необходимости вы можете заменить этот файл на свой. Для удобства я создал две отдельные процедуры `GetNumberOfIcons` и `ExtractIconsFromFile`. Первая процедура является функцией и возвращает число значков, содержащих в указанном файле. Воспользоваться этой функцией очень просто. Достаточно указать в качестве единственного параметра полный путь к файлу, и функция вернет количество значков в этом файле (листинг 19.20).

Листинг 19.20. Извлечение значков из файлов

```
'--------------------------------------
' Извлечение значков © 2004 Александр Климов
'--------------------------------------
    Declare Function ExtractIcon Lib "shell32.dll" Alias "ExtractIconA" _
         (ByVal hInst As Integer, _
         ByVal lpszExeFileName As String, _
         ByVal nIconIndex As Integer) As Integer

    ' Имя тестируемого файла
    Dim testfile As String = "c:\windows\system32\mspaint.exe"

    Private Function GetNumberOfIcons(ByVal str As String) As Integer
        Return ExtractIcon(Me.Handle.ToInt32, str, -1)
    End Function

    Private Sub butExtract_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles butExtract.Click
        ' Вычисляем число значков в файле C:\Windows\System\mspaint.exe
        lblNumberOfIcons.Text = "В файле содержится " &
        GetNumberOfIcons(testfile) & " значков"
    End Sub
```

```
' Извлекаем каждый значок из заданного файла
' выводим их на форму
Private Sub ExtractIconsFromFile(ByVal str As String)
    ' Число значков в файле
    Dim numicons As Integer
    ' Возвращаемое значение
    Dim retval As Integer
    ' Получаем число значков
    numicons = GetNumberOfIcons(str)
    ' Извлекаем первый значок
    retval = ExtractIcon(Me.Handle.ToInt32, str, 0)
    ' Присваиваем извлеченный значок свойству Icon формы
    Me.Icon = Icon.FromHandle(New IntPtr(retval))
    ' Перебираем в цикле все значки
    For i As Integer = 0 To numicons - 1
        retval = ExtractIcon(Me.Handle.ToInt32, str, i)
        Dim g As Graphics = CreateGraphics()
        ' Выводим их на форму
        g.DrawIcon(Icon.FromHandle(_
                New IntPtr(retval)), 15, i * 35 + 10)
        g.Dispose()
    Next
End Sub
```

Функция `ExtractIcon` позволяет не только подсчитывать число значков, но и извлекать их из файла. Так как функция извлекает только единственный значок за один вызов, то проходимся по всем значкам в цикле `For...Next`. Кстати, заодно я присвоил свойству `Icon` нашего окна первый извлеченный значок. Всего в файле mspaint.exe содержится шесть значков (в версии Windows XP). Бывают файлы, содержащие более двухсот значков! Теперь у вас есть возможность заглянуть вовнутрь этих файлов и просмотреть значки. Вы самый настоящий хакер! Добавляем еще одну строчку кода в обработчик события `Click` кнопки `butExtract`:

```
' Извлекаем значки
ExtractIconsFromFile(testfile)
```

19.18. Советы и рекомендации

- Попробуйте написать конвертер из чисел, записанных арабскими цифрами, в римскую систему счисления.

Глава 20

Кодируем интересно

На протяжении всей книги я рассказывал вам об интересных и занимательных примерах. А теперь я хочу обратить ваше внимание на другую сторону программирования. Ведь само написание кода тоже может стать объектом нашего исследования. Мне всегда было интересно наблюдать за стилем программирования, которого придерживаются разные люди. И часто вижу, что некоторые из них даже не подозревают о различных приемах программирования, которые могли бы облегчить написание кода и сократить время на создание программ. Особенно это относится к ветеранам Visual Basic 6.0. Перейдя на новую среду разработки, они по привычке используют старые приемы, не зная о новых возможностях VB .NET 2003. В этой главе я собрал некоторые полезные приемы, которые, несомненно, вам пригодятся в дальнейшей работе.

20.1. Мой профиль

Интегрированная среда разработки Visual Studio .NET 2003 очень гибка и легко настраиваема под разные нужды. Если вы очень сильно привыкли за долгие годы к внешнему виду и возможностям Visual Basic, то, возможно, вам стоит использовать эту схему и в VB .NET 2003. Для этого откройте на начальной странице Visual Studio ссылку **My Profile** (Мой профиль) и в списке **Profile** (Профиль) выберите строку **Visual Basic Developer**. На этой же странице вы можете настроить клавиатуру, расположение окон, параметры справки. В любой момент вы сможете сменить свой профиль. Для этого достаточно в меню **Help** (Справка) выбрать пункт **Show Start Page** (Показать стартовую страницу).

20.2. Настройка IDE

По сравнению с VB 6.0 интегрированная среда разработки (IDE) претерпела значительные изменения. Я советую посвятить один день изучению настроек программы. Я упомяну лишь несколько новых интересных нововведений.

Например, очень сильно изменились настройки текстового редактора кода. Откройте в меню **Tools** (Инструменты) пункт **Options** (Настройки) и выберите в иерархическом списке строку **Text Editor** (Текстовый редактор). Теперь вы можете выбрать автоматическую нумерацию строчек кода (флажок **Line Numbers**). Это весьма удобная настройка, особенно при написании книги. А ведь раньше приходилось устанавливать дополнительные модули от третьих фирм для достижения этой цели.

Там же, в текстовом редакторе, можно настроить автоматическое форматирование текста с отступами, как это принято у очень опытных программистов. Теперь при создании тела цикла автоматически вставляются ключевые слова для окончания цикла (я в своей практике постоянно забывал о них), и каждая новая строка цикла выводится с нужным числом отступов.

20.3. Свертывание фрагментов кода

Писать код в редакторе кода теперь одно удовольствие. Кроме привычных уже по прошлым версиям интеллектуальных подсказок, добавилась возможность свертывания фрагментов кода. Весьма удобное новшество, особенно для очень длинных проектов. Вот как это выглядит (рис. 20.1). Прежде всего, при создании новой функции или процедуры слева от введенных строк появляется знак минуса (−). Если вы щелкнете на нем мышью, то знак заменится на знак плюса (+), а текст свернется до одной строки с многоточием в конце. При подведении курсора мыши к многоточию будет показан свернутый код в отдельном всплывающем окне (рис. 20.2). Второй способ не так очевиден для большинства программистов, и они редко им пользуются. Если вы не создаете проект с нуля, а пользуетесь готовыми шаблонами, то вам доводилось видеть строчку:

```
Windows Form Designer generated code
```

Если щелкнуть на знаке плюса слева от этой строки, то откроется код, сгенерированный дизайнером форм. Можно воспользоваться этим же синтаксисом, чтобы определить свою область кода, предназначенную для сворачивания. Для этого перед требуемым блоком текста задаем регион с заданным именем:

```
#Region "Моя область сворачиваемого кода"
```

А после последней строчки указанного блока вводим завершающую инструкцию:

```
#End Region
```

В этом случае слева также появится знак минуса, при щелчке на котором вы сможете свернуть выбранный регион до одной строки. Этой возможностью

удобно пользоваться, когда вы уверены в каком-то участке кода и хотите временно скрыть его, чтоб он не мозолил глаза.

```
#Region "Моя область сворачиваемого кода"
    'В этом месте можно написать свой код
    Dim strCatName As String = "Меня зовут Рыжик"
#End Region
```

Рис. 20.1. Код в развернутом виде

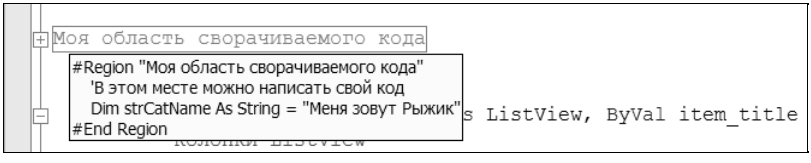

Рис. 20.2. Код в свернутом виде и отображение свернутого кода в отдельном окне

20.4. Буфер обмена

Весьма расширились возможности буфера обмена. Теперь он рассчитан на хранение сразу нескольких объектов (пользователи пакета MS Office XP уже знакомы с таким поведением буфера обмена). В процессе работы вы можете многократно копировать или вырезать код из редактора кода, и все эти строчки кода будут аккуратно храниться в специальном хранилище. Для доступа к этим строчкам достаточно выбрать вкладку **Clipboard Ring** на панели инструментов (в режиме редактирования). Я сам часто пользуюсь этой возможностью (рис. 20.3).

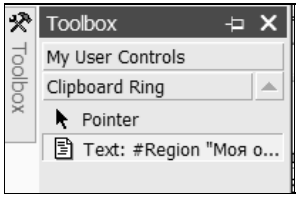

Рис. 20.3. Использование расширенного буфера обмена

20.5. Часто используемый код

Еще одна новая возможность, которой я часто стал пользоваться, — создание хранилища для часто используемого кода. Вот как это сделал я. В открытом редакторе кода надо щелкнуть правой кнопкой мыши на панели инструментов и выбрать команду **Add Tab** (можно использовать и существующую вкладку **General**, но это не совсем удобно и создаст путаницу). Задайте новое имя для созданной вкладки, например, **Часто используемый код** (рис. 20.4). Теперь просто перетащите предварительно выделенный кусок текста на эту панель. Например, если в своих программах вы постоянно используете циклы со счетчиком, то на эту панель можно поместить строку:

```
Dim counter as Integer
```

Чтобы вставить сохраненный текст в редактор, нужно дважды на нем щелкнуть, и он вставится в текущую позицию текстовой каретки. Проанализируйте свой код, и вы наверняка найдете в них повторяющиеся куски. Экономьте свое время, пользуйтесь созданной для этой цели вкладкой.

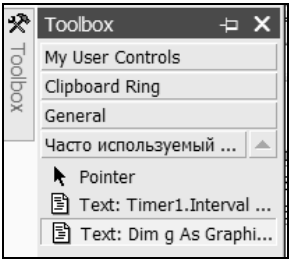

Рис. 20.4. Хранилище часто используемого кода

20.6. Управление списком задач

Относительно недавно я узнал, что Visual Basic .NET 2003 теперь поддерживает список задач — возможность, которая была раньше реализована в пакете Visual InterDev. Это очень удобная надстройка к большим проектам. Суть списка задач в следующем. В тексте программы вы создаете комментарий с ключевыми словами и описанием задач, которые надо выполнить в заданном участке кода. Перед завершающей стадией проекта можно вызвать список задач, которые вы отложили некоторое время назад, и доработать незавершенные задачи. Давайте посмотрим, как это выглядит на практике. Запустите какой-нибудь существующий проект. Введите на новой строчке

знак комментария (') и ключевое слово на выбор: `TODO`, `HACK` или `UNDONE`. После этих слов можно ввести описание комментария:

```
' TODO Проверить другие интервалы таймера
Timer1.Interval = 2000
```

Проделайте подобную операцию несколько раз в разных участках кода в разных модулях проекта. Теперь достаточно вызвать окно списка задач (командой меню **View | Other Windows | Task List** или комбинацией клавиш <Ctrl>+<Alt>+<K>), и вы сразу увидите свои закладки.

Можно задавать и пользовательские ключевые слова. В работе над книгой я применял задачу с ключевым словом `For_Book`. Для этого в меню **Tools** выбираем пункт **Options** и выделяем в иерархическом списке строку **Task List**. Введите в текстовое поле **Name** новое имя задачи (в моем случае `For_Book`), задайте ему приоритет (в случае высокого приоритета задача будет помечена восклицательным знаком) и щелкните на кнопке **Add** (Добавить). Теперь достаточно просто щелкнуть мышью на имени задачи в списке задач, и вы сразу окажетесь в нужном участке кода. Мое заключение — применять обязательно!

20.7. Шаблоны

У вас есть возможность изменить исходный код, который используется в шаблонах Visual Basic .NET при создании новых приложений. Также вы можете изменить ссылки, используемые по умолчанию в каждом проекте. Более того, вы можете создавать собственные шаблоны. Стандартные шаблоны VB .NET находятся в папке Program Files\Microsoft Visual Studio .NET 2003\Vb7\VBWizards. Внутри этой папки вы увидите множество подпапок с знакомыми именами. Например, вы можете обнаружить здесь папку WindowsApplication. Внутри этой папки имеется папка Templates\1033, которая и содержит шаблон для Windows-приложений. Вы можете отредактировать имеющиеся там файлы (например, WindowsApplication.vbproj, Form.vb и AssemblyInfo.vb) по своему вкусу.

20.8. Последовательность перехода фокуса по клавише <Tab>

Весьма упростился механизм изменения последовательности перехода фокуса элементов управления. В старой версии VB было очень неудобно менять эту последовательность, особенно при наличии большого количества элементов управления. Теперь все гораздо проще. В VB .NET появилась новая команда (**View | Tab Order**), которая упрощает задачу. После выбора этой команды все элементы управления получают порядковый номер, который

легко изменить нажатием мыши. Для отключения режима снова вызовите команду **Tab Order**.

20.9. Объявление переменных

Раньше во всех книжках по VB 6.0 предупреждали, чтобы программисты не объявляли переменные следующим образом:

```
Dim a, b, c as Integer
```

Дело в том, что по правилам синтаксиса переменной типа `Integer` становилась только последняя переменная `c`, а переменным `a` и `b` присваивался тип `Variant`. Теперь с этим покончено. И указанная выше строка работает так, как подсказывает здравый смысл, т. е. всем трем переменным присваивается тип `Integer`. Можно использовать и такую конструкцию:

```
Dim a, b as Integer, myVar, yourVar as Single
```

В этом случае переменным `a` и `b` присваивается тип `Integer`, а переменным `myVar` и `yourVar` — тип `Single`.

Также появилась возможность присвоения значения переменной при ее объявлении. Подобную возможность с нетерпением ждало сообщество программистов. Наконец-то Microsoft прислушалась к просьбам трудящихся, простите, к просьбам программистов и реализовала ее в VB .NET. Несомненно, вы будете часто использовать эту возможность в своих проектах. Если раньше приходилось писать код следующим образом:

```
Dim c as Integer
c=10
```

то теперь можно сократить код до одной строчки:

```
Dim c as Integer=10
```

Соответственно, таким образом можно объявлять и инициализировать несколько переменных:

```
Dim myVar as Integer = 10, yourVar as Double = 0.27
```

20.10. Краткая запись операций

В старых проектах на VB 6.0 для увеличения переменной на единицу применялась такая запись:

```
counter = counter + 1
```

Теперь можно использовать краткую запись:

```
counter += 1
```

Подобная запись уже давно используется программистами на C++ и Java. Естественно, что только сложением эта запись не ограничивается:

```
counter -= 1
counter /= 2
```

и т. д. В общем виде краткая запись выглядит так (табл. 20.1).

Таблица 20.1. *Краткая и полная запись*

Краткая запись	Полная запись
`A += B`	`A = A + B`
`A /= B`	`A = A / B`
`A *= B`	`A = A * B`
`A -= B`	`A = A - B`
`A ^= B`	`A = A ^ B`
`A &= B`	`A = A & B`

Я рекомендую изучить документацию по этому вопросу и привыкнуть к подобному способу кодирования. В этом случае вам будет легче читать код, написанный на других языках программирования.

20.11. Ускоренная проверка

В VB .NET 2003 появились новые ключевые слова `AndAlso` и `OrElse`. Эти операторы служат для быстрой проверки условия. Рассмотрим на примере. Пусть нам нужно проверить некоторое условие:

```
12 > 45 And MyFunction()
45 > 12 Or MyFunction()
```

В этом случае VB .NET 2003 проверяет оба условия, вызывая при этом некую функцию `MyFunction`. Но если присмотреться, то видно, что уже первая часть условия `12 > 45` является ложной, и нет смысла проверять вторую часть условия. Аналогично со второй строкой. Попробуем теперь изменить проверку следующим образом:

```
12 > 45 AndAlso MyFunction()
45 > 12 OrElse MyFunction()
```

Теперь сразу же после первой части логического условия программа прекращает проверять вторую часть и переходит к следующим операциям, не вызывая функцию. Тем самым мы экономим ресурсы компьютера, не вы-

полняя лишних операций. Подобное действие и называется *ускоренной проверкой* (short circuiting).

20.12. Создание новой функции или процедуры

Разработчики VB .NET 2003 предложили наряду со стандартным способом создания функции и другой вариант. Новый вариант больше знают программисты, знакомые с языком C++. Смысл нововведения заключается в использовании ключевого слова `Return`. Эта команда завершает вызов функции и возвращает значение, указанное после этой команды. Вот небольшой пример. Представьте себе, что вы пришли на собеседование в отдел кадров и интересуетесь размером своей будущей зарплаты в условных единицах. Вот примерный набор ваших заранее заготовленных ответов:

```
Public Function GetZarplata(ByVal Dollar As Integer) As String
    If Dollar > 1000 Then Return "Отлично"
    If Dollar > 500 Then Return "Неплохо"
    If Dollar > 200 Then Return "Ужасно"
    If Dollar > 100 Then Return "Я не желаю с вами говорить"
    Return "Не понял юмора"
End Function
```

Как видите, если кадровик предложит мне 2000 долларов, то я автоматически соглашаюсь работать.

20.13. Генератор случайных чисел

В своих примерах в данной книге я иногда использовал функцию `Rnd()`, которая перешла из VB 6.0. На самом деле использование этой функции неоправданно. На смену достаточно примитивной функции пришел мощный класс `Random`. Возможностей у нового класса гораздо больше. Вот как можно имитировать бросок игрального кубика:

```
Dim myRnD As New Random
Dim dice As Integer
dice = myRnD.Next(1, 7)
TextBox1.Text = CStr(dice)
```

Обратите внимание, что верхняя граница заданного интервала `(1, 7)` не входит в набор случайных чисел. Поэтому при бросках кубика будут выводиться только числа от 1 до 6.

20.14. Включение в решение небольшой документации

Иногда требуется написать небольшую справку к коду проекта. В принципе, можно внести документацию в виде комментариев непосредственно в сам код. Но излишняя информация засоряет проект. Гораздо лучше создать документацию в отдельном файле. Например, можно использовать гипертекстовые страницы. Тем более, что среда разработки VB .NET 2003 позволяет это сделать очень легко. Чтобы добавить HTML-страницу в решение, нужно щелкнуть правой кнопкой мыши на названии решения и в контекстном меню выбрать команду **Add | Add New Item**. В диалоговом окне выбрать шаблон **HTML Page** и нажать кнопку **Open**. В результате этих действий в решение будет добавлена пустая страница. Вы можете сразу печатать на ней, а если знакомы с языком разметки HTML-страниц, то переключитесь в режим просмотра кода и вручную пишите код с использованием тегов HTML. Находясь в любом режиме, вы можете выбрать правой кнопкой команду **View in Browser** и посмотреть, как будет выглядеть страница в вашем браузере. Включенный таким образом файл документации легко распространять вместе с проектом.

20.15. Отражение (Reflection)

А напоследок я хочу рассказать о такой вещи, как *Отражение*. Нет, здесь не имеется в виду отражение мячика от стенки. Отражение, или Reflection — это новая возможность, которая является очень важной частью .NET Framework. А интересна эта тема вот по какой причине. Возьмем, к примеру, класс `SystemInformation`. У него есть множество свойств. Предположим, вам поставили задачу написать программу, выводящую список всех свойств данного класса на экран. Вы открываете документацию и черпаете информацию о свойствах класса оттуда. Затем оформляете задачу либо через перечисление свойств с помощью цикла, либо как-то еще. Все прекрасно работает, и ваш начальник доволен. Но прошло время, и разработчики из Microsoft добавили в данный класс новые свойства. Начальство требует переписать заново программу, чтобы она учитывала эти новые добавленные свойства. Казалось бы, по-другому и быть не может. Но с появлением понятия "Отражение" стало возможным написание универсального кода! Даже с появлением новых свойств ваша программа будет прекрасно работать и выводить эти свойства на экран. За более подробной информацией об этом явлении я отсылаю вас к документации и книгам. Здесь же я покажу несколько практических примеров использования концепции отражения в проектах. В новом проекте `Reflections` расположите на форме элемент `ListView`.

Установите для него следующие свойства:

- Name = lvwInfo;
- Dock = Fill;
- View = Details.

В этом элементе будет отражаться вся необходимая информация. Также поместите на форму элемент MainMenu. Верхнему пункту меню присвойте текст Отражение, а первому подпункту — текст SystemInformation. Вот код, который выводит все доступные свойства класса SystemInformation (листинг 20.1).

Листинг 20.1. Вывод свойств

```
'--------------------------------------
' Отражение © 2004 Александр Климов
'--------------------------------------
Imports System.Reflection


    Private Sub LVWRow(ByVal lvw As ListView, ByVal item_title As String,
ByVal ParamArray subitem_titles() As String)
        ' Колонки ListView
        Dim new_item As ListViewItem = lvw.Items.Add(item_title)


        For i As Integer = subitem_titles.GetLowerBound(0) To
subitem_titles.GetUpperBound(0)
            new_item.SubItems.Add(subitem_titles(i))
        Next i
    End Sub


    Private Sub mnuSysInfo_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles mnuSysInfo.Click
        ' Заголовки колонок
        lvwInfo.Columns.Clear()
        lvwInfo.Columns.Add("Свойство", 10, HorizontalAlignment.Left)
        lvwInfo.Columns.Add("Значение", 10, HorizontalAlignment.Left)

        ' Список всех свойств класса SystemInformation
        Dim propval As Object
        Dim myType As Type = GetType(SystemInformation)
        Dim propinfo As PropertyInfo() = myType.GetProperties()
```

```
        lvwInfo.Items.Clear()
        For i As Integer = 0 To propinfo.Length - 1
            With propinfo(i)
                If .GetIndexParameters().Length = 0 Then
                    propval = .GetValue(Me, Nothing)
                    If propval Is Nothing Then
                        LVWRow (lvwInfo, _
                            .Name, _
                            "<Nothing>")
                    Else
                        LVWRow (lvwInfo, _
                            .Name, _
                            propval.ToString)
                    End If
                Else
                    LVWRow (lvwInfo, _
                        .Name, _
                        "<array>")
                End If
            End With
        Next i

        ' Устанавливаем размеры
        Dim new_wid As Integer = 30
        For i As Integer = 0 To lvwInfo.Columns.Count - 1
            lvwInfo.Columns(i).Width = -2
            new_wid += lvwInfo.Columns(i).Width
        Next i
        Me.Size = New Size(new_wid, Me.Size.Height)
    End Sub
```

Получение всех свойств класса `SystemInformation` происходит через вызов метода `GetProperties`. Получаемый массив всех свойств через метод `GetIndexParameters` выводится на экран. Как видите, мы в явном виде не используем свойства класса по их имени. Таким образом, какие бы новые свойства в будущем не появились в классе `SystemInformation`, наша программа по-прежнему будет корректно выводить все существующие свойства.

Возможности отражения гораздо шире. Рассмотрим еще один пример. Вот как можно получить все события, относящиеся к элементу управления `CheckBox` (листинг 20.2).

Листинг 20.2. Получение списка событий для элемента `CheckBox`

```
    Private Sub mnuEvents_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles mnuEvents.Click
        lvwInfo.Columns.Clear()
        lvwInfo.Columns.Add("Событие", 10, HorizontalAlignment.Left)
        lvwInfo.Columns.Add("Описание", 10, HorizontalAlignment.Left)

        ' Список событий
        Dim events_info As EventInfo() = _
            GetType(CheckBox).GetEvents()

        lvwInfo.Items.Clear()
        For i As Integer = 0 To events_info.Length - 1
            With events_info(i)
                LVWRow(lvwInfo, _
                    .Name, .ToString)
            End With
        Next i

        lvwInfo.Columns(0).Width = -2
        lvwInfo.Columns(1).Width = -2

        Dim new_wid As Integer = _
            lvwInfo.Columns(0).Width + _
            lvwInfo.Columns(1).Width + _
            30
        Me.Size = New Size(new_wid, Me.Size.Height)
    End Sub
```

Для этого примера пришлось добавить в проект еще один пункт меню `mnuEvents`. Способ получения методов практически идентичен предыдущему примеру. Только на этот раз вместо вызова метода `GetProperties` используется метод `GetEvents`. В строчке:

```
Dim events_info As EventInfo() = _
            GetType(CheckBox).GetEvents()
```

вместо `CheckBox` вы можете использовать другие классы для получения списка событий класса. Например, попробуйте использовать класс `Form1` или `MainMenu`. Чтобы окончательно закрыть эту тему, давайте попробуем теперь

получить все методы заданного класса. Как вы уже, может быть, догадались, нам понадобится вызов метода `GetMethods` (листинг 20.3).

Листинг 20.3. Получение списка методов класса `Form1`

```
    Private Sub mnuMethods_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles mnuMethods.Click
        lvwInfo.Columns.Clear()
        lvwInfo.Columns.Add("Метод", 10, _
            HorizontalAlignment.Left)
        lvwInfo.Columns.Add("Описание", 10, _
            HorizontalAlignment.Left)

        ' Список методов
        Dim methods_info As MethodInfo() = _
            GetType(Form1).GetMethods()
        lvwInfo.Items.Clear()
        For i As Integer = 0 To methods_info.Length - 1
            With methods_info(i)
                LVWRow(lvwInfo, _
                    .Name, .ToString)
            End With
        Next i

        lvwInfo.Columns(0).Width = -2
        lvwInfo.Columns(1).Width = -2

        Dim new_wid As Integer = _
            lvwInfo.Columns(0).Width + _
            lvwInfo.Columns(1).Width + _
            30
        Me.Size = New Size(new_wid, Me.Size.Height)

    End Sub
```

У изученного нами пространства имен `Reflection` есть и другие интересные и полезные возможности, которые обязательно будут встречаться в вашей практике.

20.16. Советы и рекомендации

Наверняка, у вас с годами выработался свой стиль программирования. На стиль оказывает влияние и опыт работы с другими языками программирования, и требования, предъявляемые по месту вашей работы, и ваши личные пристрастия. Но, тем не менее, стоит прислушиваться к рекомендациям, которые приводят в своих книгах опытные программисты. Стремление к некоторой стандартизации и унификации позволит вам не только лучше организовать свой процесс написания программ, но и лучше понимать код других программистов, что называется, с лету. На этом наше знакомство с примерами на VB .NET заканчивается. В *Заключении* я приведу несколько своих любимых книг и веб-страниц, которые послужили мне источниками для примеров этой книги.

Заключение

Изучение такого мощного языка, как Visual Basic .NET 2003, невозможно без постоянного самообразования путем чтения специальной литературы или просмотра в Интернете страниц признанных экспертов по выбранной теме. Здесь я хочу рассказать о своих личных пристрастиях.

Книги

- Петцольд Ч. "Программирование для Windows на Visual Basic .NET" — книга в двух томах, выпущенная издательством "Русская редакция", была переписана автором специально для VB-программистов. Эта книга полностью идентична его же книге по программированию на C#, естественно, с полной заменой листингов кода с учетом синтаксиса Visual Basic .NET. Книга написана простым и доходчивым языком, впрочем как и все работы этого автора. Множество интересных примеров не оставят равнодушным программиста любого уровня. Я не удержался и использовал несколько идей из этой книги в своих примерах.
- Гарнаев А. "Visual Basic .NET. Разработка приложений" — книга выпущена издательством "БХВ-Петербург" и содержит огромное множество небольших, но полезных и практичных примеров. Особенно хороша для начинающих программистов, которые ищут ответы на множество вопросов.

Сайты в Интернете

- **http://msdn.microsoft.com/vbasic** — "родина" Visual Basic. NET. Любой программист просто обязан посещать этот сайт, чтобы быть в курсе всех последних новостей. Примеры, обновления, интересные ссылки, документация — вот неполный перечень информации, предлагаемой сайтом.
- **http://www.bobpowell.net** — сайт Роберта Пауэлла (Robert W. Powell), на котором представлены несколько интересных советов и примеров, касающих технологии GDI+. К сожалению, часть примеров написана на C#, что создает определенные трудности для программистов, пишущих только на VB. NET 2003.
- **http://www.vb-helper.com** — регулярно обновляемый сайт Роберта Стивенса, автора многих книг по Visual Basic и Delphi. Главной особенностью

приводимых примеров является то обстоятельство, что автор предлагает коды сразу для Visual Basic 6.0 и Visual Basic .NET, что весьма удобно для тех, кто программирует на обоих языках.

- **http://www.planet-sourcecode.com** — сайт, содержащий несколько сотен тысяч строчек исходного кода, присылаемых программистами из разных стран на всеобщее обозрение.
- **http://www.codeproject.com** — на этом сайте авторы, как правило, выкладывают свой исходный код вместе со статьей, подробно описывающей предлагаемый пример. Подобный подход позволяет быстрее понять суть кода автора, чем самому изучать строка за строкой проект, содержащий сотни строк исходного кода.
- **http://www.gotdotnet.ru** — русскоязычный аналог популярного сайта **http://www.gotdotnet.com**. Здесь находится очень хороший форум, на котором участники активно задают и отвечают на вопросы, спорят, обсуждают различные проблемы.
- **http://rusproject.narod.ru** — сайт, на котором собраны наиболее интересные примеры с других сайтов, а также несколько авторских проектов.

Описание компакт-диска

На прила аемом компакт-диске помещены исходные тексты приведенных в кни е примеров. Каждый из примеров хранится в отдельном катало е с номером, соответствующим номеру лавы. Все примеры одной лавы объединены в одно решение. Для запуска отдельно о примера необходимо запустить файл решения (расширение sln) и в контекстном меню имени нужно о проекта выбрать команду **Set As StartUp**. В таблице приведено описание папок компакт-диска.

Таблица. *Описание папок компакт-диска*

Папка	Описание	Глава
/1	Проекты `FlashText`, `Matrix`, `RunTitle`, `TypingText`	1
/2	Проекты `3DText`, `CatFont`, `Credits`, `ElectricFX`, `Outline`, `RotatedText`, `RunText`, `ScrollText`, `StarWars`, `UsingPens`	2
/3	Проекты `Beziers`, `CreatePolygon`, `DrawLineSample`, `Figures`, `HatchBrushStyles`, `SolidBrushSample`, `TextureBrush2`, `TextureBrushSample`	3
/4	Проекты `Curves`, `Curves2`, `DrawCurves`, `Hipocycloid`, `Pautina`, `Spiral`	4
/5	Проекты `Balls`, `Honeycomb`, `LineraGradientBrushSample`, `PathGradientBrushSample`, `Photoshop`	5
/6	Проекты `AnalogClock`, `Matrix`, `RotatedLine`, `Shear`, `WorldTransform`	6
/7	Проекты `Fractals`, `LSystem`, `Mandelbrot`, `Serpinski`, `Stars`, `Ying-Yang`	7
/8	Проекты `Animator`, `Animator2`, `Antialias`, `ColorMatrix`, `ColorMatrixFX`, `Drag`, `DrawCube`, `MakeTransparent`, `MakeTransparent2`, `Pixel`, `RotateAngle`, `SaveImage`, `Spin`, `Warp`, `Watermark`	8
/9	Проекты `KeyboardLayout`, `KeyState`, `LockKeys`, `PrintScreen`, `SendToCalculator`	9
/10	Проекты `LinesAPI`, `LinesControlPaint`, `LinesGDIPlus`, `MovingMouse`, `Overlay`, `SimplePaint`, `Web`	10

Таблица *(окончание)*

Папка	Описание	Глава
/11	Проекты `ActiveForm`, `DisableX`, `DragForm`, `FlashWindow`, `RodStephens_Ssaver`, `Simple_Ssaver`, `SystemMenu`, `TextForm`, `TitleBarClick`, `TransparentForm`	11
/12	Проекты `AniLabel`, `AutoClose`, `BitmapListView`, `BitmapMenu`, `BlinkNotify`, `ColorMainMenu`, `ColorMenu`, `ColorStatusBar`, `DigitTextbox`, `EnablesStyles`, `Manifest`, `NumberLines`, `RadiobuttonSamples`, `RedCheckbox`, `Resource`, `RoundControls`, `StyleXP`, `TextBoxSamples`, `TextContextMenu`	12
/13	Проекты `AI`, `BlockAll`, `HappyNewYear`, `MadMouse`, `Mind`, `Mouse`, `MouseTrap`, `OpenCD`, `Pusk`, `RunawayButton`, `SpyCat`, `TrayClockReplace`, `TwoButtons`, `TypingMachine`	13
/14	Проекты `3DShapes`, `Orbison`	14
/15	Проекты `MP3`, `PlaySound`, `VBPiano`, `Winamp`	15
/16	Проекты `MSAgent`, `SAPI5`, `TwoAgents`	16
/17	Проекты `Bounce`, `Bounce2`, `FallBall`, `RealBall`	17
/18	Проекты `Cards`, `Safe`, `Safe2`, `Snake`, `TicTacToe`, `XO`	18
/19	Проекты `AniCursor`, `Caret`, `ColorConsole`, `EasterEgg`, `ExtractIcons`, `FileProperties`, `Friday`, `GetScreenShot`, `OneInstance`, `OpenPass`, `RightCorner`, `Roman2Arab`, `RunApp`, `Version`, `Wallpaper`, `WebSendMail`, `WinVersion`	19
/20	Проект `Reflections`	20

Предметный указатель

Д

З

И

К

Л

М

(окончание рубрики см. на с. 512)

P

C

Т

У

Ф

Х

Ц

Ч

Ш

Э

Я